The R Journal

Volume 8/1, Aug. 2016

A peer-reviewed, open-access publication of the
R Foundation for Statistical Computing

Contents

Editorial 4

Contributed Research Articles

metaplus: An R Package for the Analysis of Robust Meta-Analysis and Meta-Regression 5

Gender Prediction Methods Based on First Names with **genderizeR** 17

Conditional Fractional Gaussian Fields with the Package **FieldSim** 38

rTableICC: An R Package for Random Generation of 2×2×K and R×C Contingency Tables. 48

Maps, Coordinate Reference Systems and Visualising Geographic Data with **mapmisc** 64

Variable Clustering in High-Dimensional Linear Regression: The R Package **clere** . . 92

Stylometry with R: A Package for Computational Text Analysis 107

quickpsy: An R Package to Fit Psychometric Functions for Multiple Groups 122

FWDselect: An R Package for Variable Selection in Regression Models 132

An Interactive Survey Application for Validating Social Network Analysis Techniques 149

Exploring Interaction Effects in Two-Factor Studies using the **hiddenf** Package in R. . 159

Heteroscedastic Censored and Truncated Regression with **crch** 173

Model Builder for Item Factor Analysis with **OpenMx** 182

Spatio-Temporal Interpolation using **gstat** 204

SWMPr: An R Package for Retrieving, Organizing, and Analyzing Environmental Data for Estuaries 219

CryptRndTest: An R Package for Testing the Cryptographic Randomness 233

scmamp: Statistical Comparison of Multiple Algorithms in Multiple Problems 248

keyplayer: An R Package for Locating Key Players in Social Networks 257

SchemaOnRead: A Package for Schema-on-Read in R 269

Crowdsourced Data Preprocessing with R and Amazon Mechanical Turk 276

mclust 5: Clustering, Classification and Density Estimation Using Gaussian Finite Mixture Models 289

clustering.sc.dp: Optimal Clustering with Sequential Constraint by Using Dynamic Programming 318

progenyClust: an R package for Progeny Clustering 328

statmod: Probability Calculations for the Inverse Gaussian Distribution 339
Using **DECIPHER** v2.0 to Analyze Big Biological Sequence Data in R 352
R Packages to Aid in Handling Web Access Logs 360
Nonparametric Tests for the Interaction in Two-way Factorial Designs Using R 367
GMDH: An R Package for Short Term Forecasting via GMDH-Type Neural Network Algorithms 379
sbtools: A Package Connecting R to Cloud-based Data for Collaborative Online Research 387

News and Notes

Conference Report: useR! 2016 399
Changes on CRAN 402
News from the Bioconductor Project 404
Changes in R 406

The R Journal is a peer-reviewed publication of the R Foundation for Statistical Computing. Communications regarding this publication should be addressed to the editors.

Prospective authors will find detailed and up-to-date submission instructions on the Journal's homepage.

R Journal Homepage:
http://journal.r-project.org/

Email of editors and editorial board:
firstname.lastname@R-project.org

The R Journal is indexed/abstracted by EBSCO, DOAJ, and Thomson Reuters.

Editorial

by Michael Lawrence

On behalf of the editorial board, I am pleased to publish Volume 8, Issue 1 of the R Journal. This issue contains 27 contributed research articles. Each of them either presents an R package, a specific extension of an R package or applications using R packages available from the Comprehensive R Archive Network (CRAN, http:://CRAN.R-project.org). It thus provides a small but current cross-section of the burgeoning R ecosystem.

Interest in developing graphical user interfaces and visualization tools on top of R, and integrating R with the web, continues to grow, as evidenced by the articles on the Social Network Analysis Survey Framework, a Shiny interface to the OpenMX modeling software, and the mapmisc package for visualizing geographic data. This issue also includes articles on R interfaces to cloud-based data resources (the sbtools package), and a system for crowd-sourcing data preprocessing chores (the MTurkR package).

True to the roots of R, the bulk of this issue presents advancements in the field of applied statistics, including the crch package for modeling censored and truncated data, new improvements in the mclust package for fitting Gaussian mixture models, the scmamp package for comparing the performance of multiple algorithms, the rTableICC for randomly generating contingency tables, the clere package for variable clustering in high dimensions, the FWDselect package for forward model selection, the metaplus package for analyzing robust meta-analyses, the hiddenf package for exploring interaction effects in factorial studies, the statmod package for calculating probabilities with the inverse Gaussian distribution, the clustering.sc.dp package for clustering with sequential constraints and a review of R-based methods for non-parametric testing of interactions in two-way factorial designs.

The diversity of the R ecosystem is such that packages are available for many highly focused subfields. Examples in this issue include the stylo package for performing stylometry studies, the CryptRndTest package for analyzing randomness in cryptography, the quickpsy package for function fitting in psychometrics, SWMPr for analyzing estuary data, FieldSim for simulating Gaussian fields (e.g., in image analysis), progenyClust for progeny clustering, keyplayer for finding key players in social networks, DECIPHER for deciphering biological sequence data, GMDH for short term forcasting with neural networks, and gstat for spatio-temporal interpolation of geostatistics data.

Before the user can apply these tools, the data must first be imported into R and munged into a shape that is amenable to analysis. We present several packages for importing and munging data, namely: SchemaOnRead, a generalized data import framework supporting numerous common file types, multiple packages for working with web logs (webreadr, urltools, iptools and rgeolocate), and the genderizeR package for predicting gender from first names.

In addition the News and Notes section contains the usual updates on CRAN and the Bioconductor project.

I hope you enjoy the issue.

Michael Lawrence
Michael.Lawrence@r-project.org

metaplus: An R Package for the Analysis of Robust Meta-Analysis and Meta-Regression

by Ken J. Beath

Abstract The **metaplus** package is described with examples of its use for fitting meta-analysis and meta-regression. For either meta-analysis or meta-regression it is possible to fit one of three models: standard normal random effect, *t*-distribution random effect or mixture of normal random effects. The latter two models allow for robustness by allowing for a random effect distribution with heavier tails than the normal distribution, and for both robust models the presence of outliers may be tested using the parametric bootstrap. For the mixture of normal random effects model the outlier studies may be identified through their posterior probability of membership in the outlier component of the mixture. Plots allow the results of the different models to be compared. The package is demonstrated on three examples: a meta-analysis with no outliers, a meta-analysis with an outlier and a meta-regression with an outlier.

Introduction

Meta-analysis is a method of combining the results of different studies to produce one overall result (Sutton et al., 2000). Meta-regression is an extension to meta-analysis which allows study-specific effect sizes to change depending on study-specific covariates. For example there may be studies comparing a drug to placebo, with varying doses of the drug used in the different studies. It is possible that the effectiveness of the drug will vary with dose, in a linear or nonlinear relationship, and by including this in the model the unexplained variation is reduced.

One of the difficulties in combining studies is that the differences between studies may be greater than would be indicated by the variation within each study. This is allowed for by the random effect model where the effect for each study has two components: an overall effect and a random component specific to each study, with the random component traditionally assumed to have a normal distribution. The model without a random effect is known as the fixed effect model, which is equivalent to a random effect model with zero variance for the random effect.

One difficulty is that the assumption of a normally distributed random effect may be unrealistic, with a particular violation that the tails are heavier than would be expected. While it has been shown that results are robust to moderate violations of the normality assumption (Kontopantelis and Reeves, 2012), this does not apply to more extreme cases. One solution to this is to use an alternative to the normal distribution for the random effect, for example the *t*-distribution, as described in Lee and Thompson (2008) and Baker and Jackson (2008), the Laplace distribution (Demidenko, 2013, Section 5.1.5), a non-parametric (Branscum and Hanson, 2008) or a semi-parametric (Burr and Doss, 2005) random effect distribution. This, however, does not identify which studies are unusual. A traditional method of identifying outliers is through residual diagnostics and this has been applied to meta-analysis by Viechtbauer and Cheung (2010). However, the effect of the outliers on the fitted model may cause them to be masked (Atkinson, 1986). This occurs when the outliers affect the fitted model to the extent that the unusual observations no longer appear unusual. A method to avoid this is deletion of residuals, used in Viechtbauer and Cheung (2010), but this is only effective for single outliers. It can be extended to allow multiple outliers but with the need to fit a large number of models. A method to avoid the problem of multiple outliers is described by Gumedze and Jackson (2011). They assume that studies are either normal or are outlier studies from a random effect distribution with a higher variance. Only one study is assumed to be an outlier, with each study tested in turn, but multiple outliers then allowed for using order statistics. Beath (2014) noted the similarity of this model to a mixture model, which also allows for a more general fitting algorithm and a statistical test for the presence of outliers and indication of which studies are outliers.

The purpose of the **metaplus** package (Beath et al., 2016) is to fit the two robust models with random effects based on the *t*-distribution and the mixture of normals, as well as the standard normal random effects model. It is not designed to replace a more general meta-analysis package, such as **metafor** (Viechtbauer, 2010) but to provide additional specialised analyses. In producing forest plots, it builds upon the functionality of the **metafor** package, allowing the various models to be compared.

Models

The random effect meta-analysis model assumes that the observed treatment effect Y_i for study i is

$$Y_i = \mu + E_i + \epsilon_i,$$

where μ is the overall mean for the studies, E_i is a random effect with mean zero, and ϵ_i is a normally distributed error with variance σ_i^2 for study i, where the within study variance σ_i^2 is assumed to be known.

An extension, known as meta-regression, to the random effect meta-analysis model is to include covariates to explain the heterogeneity (Sutton et al., 2000, p. 51). Incorporating this into the meta-analysis model we obtain

$$Y_i = \mu + X_i^T \beta + E_i + \epsilon_i,$$

where X_i is a vector of covariate values for study i, and β is a vector of the corresponding parameters.

In **metaplus** there are three available random effect distributions:

Normal: The probability density function for study i is

$$f\left(Y_i|X_i;\mu,\tau\right) = \frac{1}{\sqrt{2\pi\left(\sigma_i^2+\tau^2\right)}} \exp\left(-\frac{\left(Y_i-\mu-X_i^T\beta\right)^2}{2\left(\sigma_i^2+\tau^2\right)}\right).$$

Robust t-distribution: This distribution was introduced as one of a number of distributions for robust meta-analysis by Lee and Thompson (2008) and Baker and Jackson (2008). This approach replaces the normal random effect distribution with a t-distribution. The degrees of freedom (ν) of the t-distribution control the heaviness of the tails, and are estimated from the data, using ν^{-1} as the parameter for numerical advantages. The probability density function no longer has a closed-form expression, requiring integration over the t-distribution random effect as

$$f\left(Y_i|X_i;\mu,\tau,\nu\right) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \int_{-\infty}^{\infty} \exp\left(\frac{-\left(Y_i-\mu-X_i^T\beta-\eta\right)^2}{2\sigma_i^2}\right) g\left(\eta|\tau,\nu\right) d\eta,$$

where $g\left(\eta|\tau,\nu\right)$ is the density function of a scaled t-distribution with ν degrees of freedom

$$g\left(\eta|\tau,\nu\right) = \frac{\Gamma\left(\left(\nu+1\right)/2\right)}{\tau\sqrt{\pi\nu}\Gamma\left(\nu/2\right)}\left(1+\frac{\eta^2}{\nu\tau^2}\right)^{-\left(\left(\nu+1\right)/2\right)}.$$

Robust mixture: This assumes that a study can belong to one of two classes, where each class is a standard random effect model with the same mean but different random effect variance, which is higher for the outlier class (Beath, 2014). The robust meta-regression model takes the form

$$Y_{i|k} = \mu + X_i^T \beta + E_{i|k} + \epsilon_i,$$

where ϵ_i is as for the standard model, but $E_{i|k}$ is now a random effect dependent on the class, where $k = 1, 2$ indexes the classes, with $k = 1$ corresponding to standard studies and $k = 2$ to outlier studies, with random effect variances τ_1^2, τ_2^2 respectively, with the restriction that $\tau_2^2 > \tau_1^2$, and again zero mean. The probability density function becomes the weighted sum of the probability density function for each class, with weights equal to the proportion of studies in each class π_1, π_2 for the standard and outlier studies, respectively:

$$f\left(Y_i|X_i;\mu,\tau_1,\tau_2,\pi_1,\pi_2\right) = \sum_{k=1}^{2} \pi_k \frac{1}{\sqrt{2\pi}}\left(\frac{1}{\sigma_i^2+\tau_k^2}\right)^{1/2} \exp\left(-\frac{1}{2}\frac{\left(Y_i-\mu-X_i^T\beta\right)^2}{\sigma_i^2+\tau_k^2}\right)$$

with the constraints that $\pi_1 + \pi_2 = 1$ and $0 \leq \pi_i \leq 1$.

Profile likelihood based confidence intervals

A difficulty with the use of standard maximum likelihood techniques for random effect models is that they produce biased estimates for the variance of the random effect, which results in biased estimates of the standard errors for the parameters of interest, and therefore poor coverage using Wald-type confidence intervals. The solution for meta-analysis has been the use of Wald-type confidence intervals

obtained from models fitted using residual maximum likelihood (REML), but this is difficult for the robust models. However, profile likelihood based confidence intervals (Pawitan, 2001, p. 61) have been found to be superior (Hardy and Thompson, 1996), and these are used for all fitted models. The profile likelihood based confidence intervals are obtained from routines based on the `mle2` function in the package **bbmle** (Bolker and R Core Team, 2014) which provides an extended version of `mle`. The p-values are calculated using the likelihood ratio test statistic so that they are consistent with the confidence intervals.

Parametric bootstrap

Testing for the need for the robust distributions requires a test of $\nu = \infty$, or equivalently $\nu^{-1} = 0$ for the t-distribution and $\pi_2 = 0$ for the robust mixture. Both tests involve a test of a parameter on the boundary of the parameter space, so the usual asymptotic theory cannot be used. One solution is the parametric bootstrap (McLachlan, 1987), which involves simulating data sets under the null hypothesis and calculating the likelihood ratio test statistic for each simulated data set. The observed test statistic is then compared to the simulated test statistics to determine the p-value.

Other computational details

For both robust models the starting values are important, as the optimisation used to obtain the maximum likelihood may converge to a local minimum. For the t-distribution a standard normal random effect model is first fitted. The parameter estimates from this model together with a range of values of the t-distribution degrees of freedom are used as starting parameter values for the t-distribution random effect model. From these fitted models the model with the maximum likelihood is chosen as the final fitted model.

For the t-distribution random effect model numerical integration is used to obtain the marginal likelihood, with a choice of either adaptive quadrature or adaptive Gauss-Hermite quadrature. In general adaptive quadrature was found to be superior; however it was required to use adaptive Gauss-Hermite quadrature when the standard errors of studies are unusually small. Another difficulty is that the model is not identifiable when $\tau^2 = 0$ as the likelihood is no longer dependent on the t-distribution degrees of freedom, and this causes difficulties with the optimisation. To avoid this a model was fitted with $\nu^{-1} = 0$, to allow $\tau^2 = 0$, and the likelihood from this model used if it was equal or larger than given by the optimisation with ν unconstrained.

For the robust mixture model a generalized EM (GEM) algorithm is used. The usual method for generating starting values for a mixture model using the EM algorithm, as described in McLachlan and Peel (2000, p. 55), is to randomly allocate subjects to each group in the initial E step. This is repeated for a number of random allocations and the resulting model fit with the highest maximum likelihood used as the fitted model. For the outlier models this usually requires a large number of random allocations, and therefore model fits, due to the small number in the outlier class.

The method used in **metaplus** is to systematically generate the initial outliers in the E step with an increasing number of initial outliers, starting with no outliers. For a given number of outliers in the selected initial set all possible initial sets are fitted with the restriction that each set of initial outliers builds on the best set of initial outliers found for the previous number of outliers. For example if, when considering single initial outliers, study 10 as the initial outlier produces the highest maximum likelihood then study 10 would be included in all pairs of studies when considering models with two initial outliers. When the maximum likelihood does not increase the process is stopped.

Using package metaplus

The main function available in **metaplus** is `metaplus`, with associated methods `outlierProbs` and `testOutliers` specific to **metaplus**, with the arguments for each shown in Table 1. The function `metaplus` fits a meta-analysis model to the studies, with results extracted using `summary`, and plotted using `plot`. The `plot` method makes use of the `forest` method in **metafor** allowing the same customisations of the plots. An additional argument specific to `plot` in **metaplus** is `extrameta`, which allows for extra meta-analysis results to be plotted. This allows for different models (i.e. standard and robust) to be compared, or for meta-regression to show the overall effect at different values of the covariates. An alternative method of plotting is to use **forestplot** (Gordon and Lumley, 2015) which allows some other customisations, but will require combining the data from the studies and summaries. The method `testOutliers` tests for the presence of outliers for the robust models using the parametric bootstrap. The method `outlierProbs` determines the posterior probability of each

	`metaplus()` arguments
`yi`	Vector of observed effect sizes corresponding to each study.
`sei`	Vector of observed standard errors corresponding to each study.
`mods`	Data frame of covariates corresponding to each study (only required for a meta-regression model).
`random`	The type of random effect distribution. One of `"normal"`, `"t-dist"`, `"mixture"`, for standard normal, t-distribution or mixture of normals, respectively.
`label`	The label to be used for this model when producing the summary line on the forest plot. This allows for identification of the model when comparing multiple models.
`plotci`	Should a diagnostic plot for the profile likelihood be made? See the package **bbmle** documentation for further details.
`justfit`	Should the model only be fitted? If only the model is fitted then profiling and likelihood ratio test statistics are not calculated. This is useful for bootstrapping to reduce computation time.
`slab`	Vector of character strings corresponding to each study. This is used only to label the plots.
`useAGQ`	Should adaptive Gauss-Hermite quadrature be used with the t-distribution random effect model. This may be used when there are numerical problems due to small standard errors.
`quadpoints`	Number of quadrature points for the adaptive Gauss-Hermite quadrature.
`data`	Optional data frame in which to search for other variables.

	`outlierProbs()` arguments
`object`	"metaplus" object.

	`testOutliers()` arguments
`object`	"metaplus" object.
`R`	Number of simulations used in the parametric bootstrap.

Table 1: Arguments for functions and methods of the **metaplus** package.

study being an outlier for the normal mixture model. The returned object has an associated `plot` method to plot the outlier probabilities. The returned results are shown in Table 2.

Examples

In the following examples, both robust options are used to demonstrate the capabilities of the package. In practice it will be required to choose which model to use when determining the final result. This should be the better fitting model, which can be determined using either AIC or BIC. Where the outliers are extreme the t-distribution will fit poorly requiring the use of the mixture distribution. In other cases the t-distribution will be preferred as it uses one less parameter, also making it less likely to produce unstable results which will be shown in the confidence interval profile plot. Where there is little difference between the fits the mixture distribution may be preferred as it allows identification of the outlier studies.

Intravenous magnesium in acute myocardial infarction

A number of studies have been performed to determine the effectiveness of intravenous magnesium in acute myocardial infarction, and a meta-analysis is performed in Sterne et al. (2001). The studies have caused considerable controversy, as the results of a single large study ISIS-4 (ISIS-4: Collabarative Group, 1995) contradicts the results of a meta-analysis. Higgins and Spiegelhalter (2002) discuss some of the history and some suggested methods from a Bayesian perspective, Woods (2002) comments on the variability between studies due to timing of infusion, and Downing (1999) on the higher level of dose used in ISIS-4, with a more recent meta-analysis by Li et al. (2009). Of interest is whether,

	`metaplus()`
`results`	Matrix containing columns for estimate, lower and upper 95% confidence interval and p-value. If `justfit = TRUE` then only the parameter estimates are returned.
`yi`	Vector of observed effect sizes.
`sei`	Vector of observed standard errors corresponding to each effect size.
`mods`	Data frame of covariates corresponding to each study (only returned from a meta-regression model).
`fittedmodel`	Final model returned from **bbmle**.
`justfit`	Value of `justfit` passed to `metaplus`.
`random`	Type of random effect.
`slab`	Vector of character strings corresponding to each study. This is used to label the forest plot.

	`outlierProbs()`
`outlier.prob`	Vector of posterior probabilities that the study is an outlier corresponding to each study.
`slab`	Vector of labels for the studies.

	`testOutliers()`
`pvalue`	p-value obtained from the parametric bootstrap.
`observed`	Observed value of the likelihood ratio test statistic.
`sims`	Vector of simulated values of the test statistic under the null hypothesis.

Table 2: Results reported by functions and methods of the **metaplus** package.

given the heterogeneity between studies, the ISIS-4 study is unusual. The data have been obtained in the form of log odds ratios for mortality where negative values correspond to treatment benefit, but if raw data in the form of number of events per number of patients is available, then these can be converted using, for example, the `escalc` function in the **metafor** package. The standard random effect meta-analysis can be performed, and the parameter estimates obtained as follows:

```
> mag.meta <- metaplus(yi, sei, slab = study, data = mag)
> summary(mag.meta)

         Est. 95% ci.lb 95% ci.ub   pvalue
muhat -0.7463   -1.2583   -0.3428 0.000501
tau2   0.2540

     logLik      AIC      BIC
  -19.68459 43.36918 44.91436
```

Adding the argument `plotci = TRUE` will produce a plot giving details of the profile confidence intervals, as shown in Figure 1. The basis of the plot is that the profile log likelihood in the region of the maximum likelihood estimate should be asymptotically quadratic. As differences from a quadratic are difficult to determine by eye, a transformation is performed to the z scale, so that the curve should follow a straight line. Rather than plotting z, $|z|$ is plotted so that the curve should then be in the form of a symmetric "V" (Bolker and R Core Team, 2014). In this case, the shape is not symmetric, so this does not hold, although the difference is not large enough to be important. This is confirmed by the lack of symmetry of the confidence interval for `muhat`. An important variation from the "V" occurs when either half of the curve may not be monotonic, indicating that the profile likelihood is multi-modal and if this occurs in a region affecting the confidence interval then the calculated confidence interval may be incorrect. It may also be an indication that the model used is incorrect or that there is insufficient data for the fitted model.

The forest plot showing the studies and overall effect can be obtained using `plot(mag.meta)`. The **metaplus** package uses the forest plot capabilities of the **metafor** package which allows the arguments for the `forest` plot in **metafor** to be used when plotting. As the results for the magnesium studies are log odds ratios it is more useful to produce plots with units of odds ratios. This can be obtained by annotating the horizontal axis with odds ratios corresponding to the log odds, and requesting

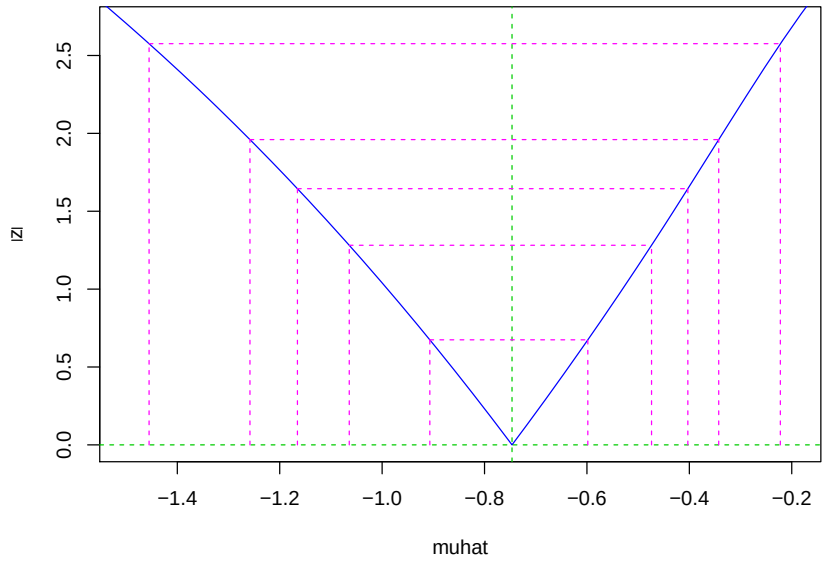


Figure 1: Profile plot for intravenous magnesium in acute myocardial infarction using the normal random effect model.

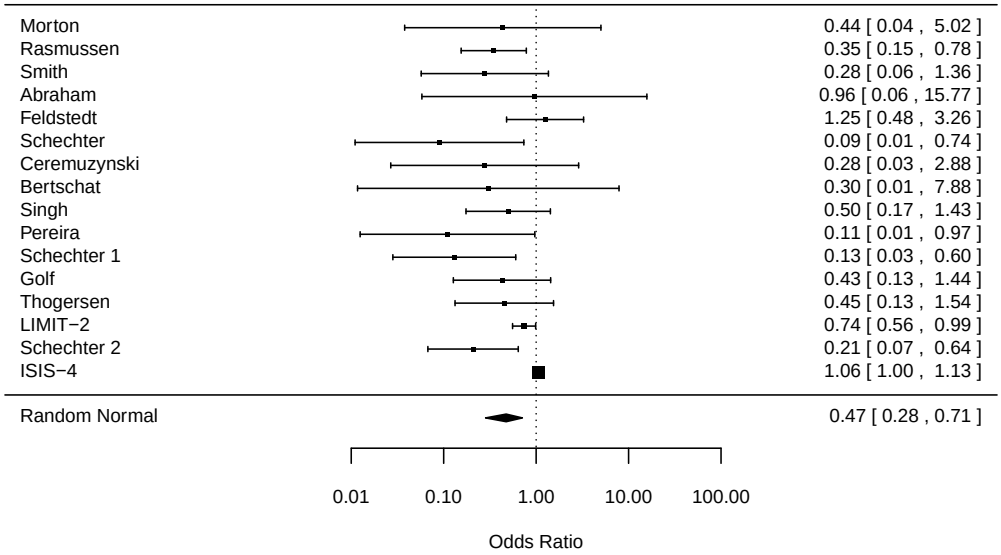


Figure 2: Forest plot for magnesium studies for mortality using the normal random effect model.

an exponential transformation for the coefficients, as shown in the following code, and the plot is shown in Figure 2. The documentation for the **metafor** package should be investigated for further modifications. Under some systems the characters will not be properly spaced. This can be solved by using the **extrafont** (Chang, 2014) package and a fixed width font, for example 'Courier New'.

```
> plot(mag.meta, atransf = exp, at = log(c(.01, .1, 1, 10, 100)),
+   xlab = "Odds Ratio", cex = 0.75)
```

The meta-analysis is repeated using a *t*-distribution for the random effect by adding the `random = "t-dist"` argument. From the summary the estimate of `vinv`, the inverse degrees of freedom, is zero corresponding to infinite degrees of freedom, or a normal distribution. The BIC is also a guide, with an increase for the *t*-distribution model indicating that a standard normal is the correct model.

```
> mag.tdist <- metaplus(yi, sei, slab = study, random = "t-dist", data = mag)
> summary(mag.tdist)

         Est. 95% ci.lb 95% ci.ub   pvalue
muhat -0.7463   -1.2583   -0.3430 0.000501
tau2   0.2540
```

```
vinv   0.0000

     logLik      AIC      BIC
  -19.68459 45.36918 47.68695
```

This can be confirmed with the `testOutliers` command, which performs a parametric bootstrap to obtain the null distribution of the likelihood ratio test statistic for the test that $\nu^{-1} = 0$, required as the test of the parameter is on the boundary of the parameter space. Note that this may take some time for the default of 999 simulations, of the order of one hour or longer depending on the number of studies, so initial investigation may be performed with a smaller number of simulations, with consequently lower accuracy.

```
> summary(testOutliers(mag.tdist))

Observed LRT statistic 0.0 p value 1
```

The analysis can be repeated using the robust mixture distribution for the random effect. The variance of both the random effect for standard studies (`tau2`) and for outlier studies (`tau2out`) are very close indicating that there are no outlier studies and this is confirmed by the outlier test.

```
> mag.mix <- metaplus(yi, sei, slab = study, random = "mixture", data = mag)
> summary(mag.mix)

                    Est.  95% ci.lb  95% ci.ub   pvalue
muhat         -0.7463147 -1.2593989 -0.3427085 0.000777
tau2           0.2539981
tau2out        0.2540892
Outlier prob.  0.0001904

     logLik      AIC      BIC
  -19.68459 47.36918 50.45954

> summary(testOutliers(mag.mix))

Observed LRT statistic 0.0 p value 1
```

CDP choline for cognitive and behavioural disturbances

This meta-analysis evaluates the effect of CDP choline for cognitive and behavioural disturbances associated with chronic cerebral disorders in the elderly (Fioravanti and Yanagi, 2005) using standardised mean differences of memory measures as the outcome. A study (Bonavita 1983) was previously determined to be an outlier by Gumedze and Jackson (2011). A standard random effect meta-analysis will be fitted first, as previously.

```
> cdp.meta <- metaplus(yi, sei, slab = study, data = cdp)
> summary(cdp.meta)

        Est. 95% ci.lb 95% ci.ub pvalue
muhat 0.38944   0.07269   0.76634 0.0218
tau2  0.14666

     logLik      AIC      BIC
  -8.198544 20.39709 21.00226
```

A robust model using the t-distribution is fitted with the following code.

```
> cdp.tdist <- metaplus(yi, sei, slab = study, random = "t-dist", data = cdp)
> summary(cdp.tdist)

          Est. 95% ci.lb 95% ci.ub  pvalue
muhat 1.946e-01 5.296e-02 3.610e-01 0.00899
tau2  4.478e-05
vinv  2.024e+00

     logLik      AIC      BIC
  -4.057683 14.11537 15.02312

> summary(testOutliers(cdp.tdist))
```

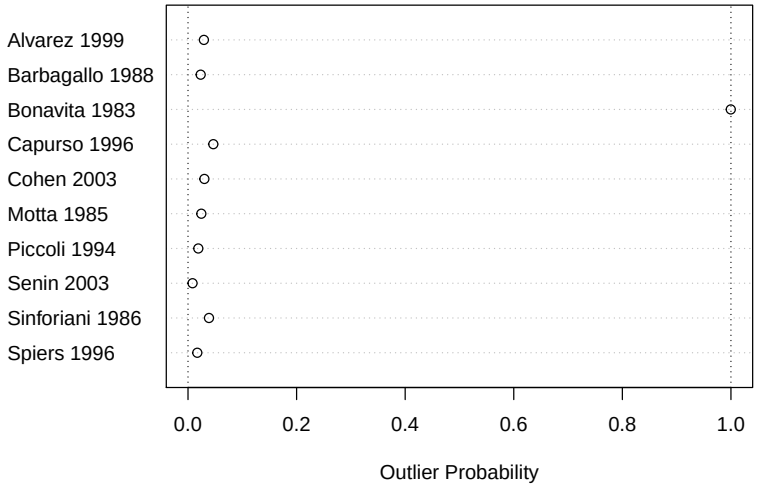


Figure 3: Outlier probabilities for CDP studies from the robust mixture random effect model.

```
Observed LRT statistic 8.3 p value 0.001
```

As a rough guide, the decrease in AIC and BIC demonstrates that the model is an improvement, and this is confirmed with the outlier test. The fit is repeated using the robust mixture.

```
> cdp.mix <- metaplus(yi, sei, slab = study, random = "mixture", data = cdp)
> summary(cdp.mix)

                Est. 95% ci.lb 95% ci.ub  pvalue
muhat         0.1910    0.0563    0.3479 0.00711
tau2          0.0000
tau2out       3.1558
Outlier prob. 0.1237

     logLik      AIC      BIC
  -3.007145 14.01429 15.22463

> summary(testOutliers(cdp.mix))

Observed LRT statistic 10.4 p value 0.001
```

The output from the robust mixture model has an interesting feature. For standard studies the estimated random effect variance is zero, indicating that only the outlier studies are contributing to the heterogeneity. The posterior probability of each study being an outlier can be obtained as:

```
> cdp.mix.outlierProbs <- outlierProbs(cdp.mix)
```

and plotted using `plot(cdp.mix.outlierProbs)` in Figure 3. This shows clearly that Bonavita 1983 has a posterior probability of nearly 1.0 of being an outlier. The other studies have a non-zero posterior probability of being outliers, as there is an overlap between the distribution of the standard and outlier studies, but are relatively close to zero.

Lastly, a forest plot with the results of all three models is generated, using the `extrameta` parameter to add the robust models, i.e. `plot(cdp.meta,extrameta = list(cdp.tdist,cdp.mix))`, and these are shown in Figure 4, where it can be noted that Bonavita 1983 has an unusually high value. The effect of the robust models is to down-weight the Bonavita 1983 study, which has the consequence of both reducing the overall effect estimate and its standard error.

Exercise for depression

This example is a meta-analysis of trials of exercise in the management of depression (Lawlor and Hopker, 2001). Higgins and Thompson (2004) used the data as an example of meta-regression using a number of covariates, which will be limited here to a single covariate, the duration of trial. The outcome is effect size calculated using Cohen's method. First the meta-analysis using standard normal random effect and the robust mixture model are performed. The data will be ordered by duration to assist in identifying a variation from the linear relationship.

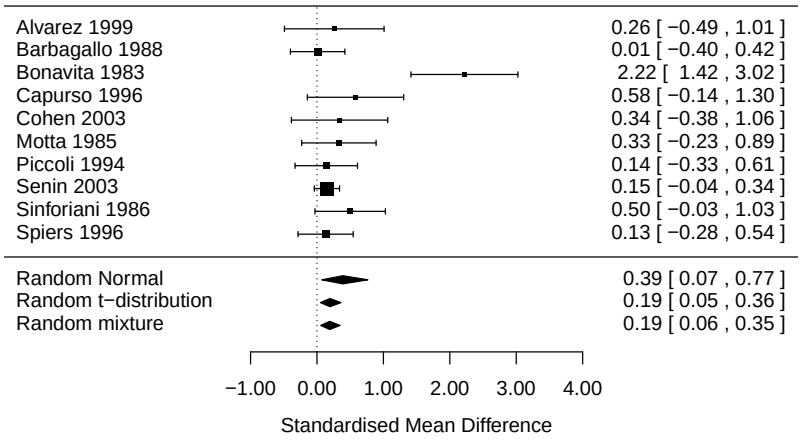


Figure 4: Forest plot for CDP studies (standardised mean difference for memory measures) with summaries.

```
> exercise <- exercise[order(exercise$duration), ]
> exercise.meta <- metaplus(smd, sqrt(varsmd), mods = duration, slab = study,
+   data = exercise)
> summary(exercise.meta)

            Est. 95% ci.lb 95% ci.ub   pvalue
muhat    -2.8994   -4.3006   -1.5222 0.000884
tau2      0.1171
duration  0.2078    0.0584    0.3632 0.011570

     logLik      AIC      BIC
  -8.133435 22.26687 23.17462

> exercise.mix <- metaplus(smd, sqrt(varsmd), mods = duration, slab = study,
+   random = "mixture", data = exercise)
> summary(exercise.mix)

                 Est. 95% ci.lb 95% ci.ub   pvalue
muhat         -2.88472  -4.11082  -1.48262 0.000649
tau2           0.00000
tau2out        0.59398
Outlier prob.  0.25169
duration       0.21086   0.07808   0.34586 0.007052

    logLik      AIC     BIC
  -7.69139 25.38278 26.8957

> exercise.testOutliers <- testOutliers(exercise.mix)
> summary(exercise.testOutliers)

Observed LRT statistic 0.9 p value 0.075

> exercise.outlierProbs <- outlierProbs(exercise.mix)
```

The test for outliers was close to being significant ($p = 0.075$); however a conservative approach seems appropriate, by using the robust model where the presence of outliers is not conclusive but there is a reasonable amount of evidence that there are outliers, as in this case. Note also that the p-value is different from that obtained in Beath (2014), due to the use of randomly generated data in the parametric bootstrap. Running the parametric bootstrap with a large number of simulations showed that the p-value was actually near 0.04. Using `plot(exercise.outlierProbs)` the outlier probabilities are shown in Figure 5 where the study by Reuter is an obvious outlier with a posterior probability greater than 0.9. This study is a dissertation and was not published in a peer-reviewed journal, and was not included in a later meta-analysis by Krogh et al. (2011). There is also strong evidence of the effect of trial duration.

As **metaplus** does not currently have a `predict` method, the alternative to calculate the effect at each of Weeks 4, 8 and 12 is to centre the data at those times and fit a meta-regression for each (Johnson

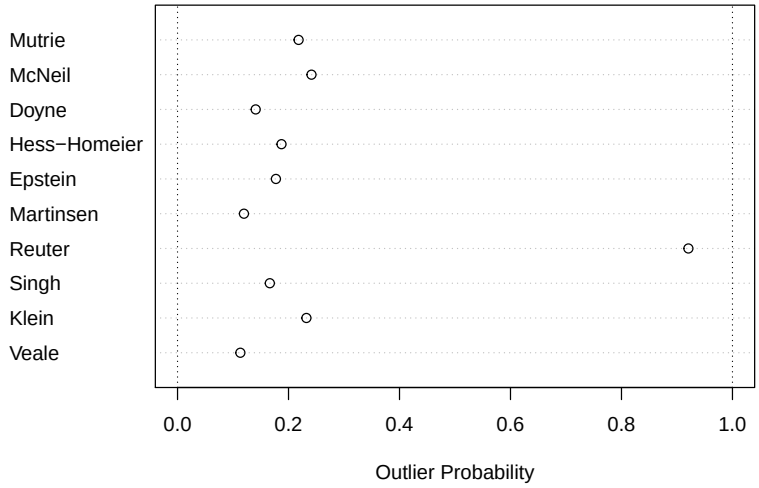


Figure 5: Outlier probabilities for depression versus exercise from the robust mixture random effect model.

and Huedo-Medina, 2011). The intercept for each meta-regression will then be the estimated mean effect at that time. A model without including the covariate for study duration is also fitted. The forest plot is shown in Figure 6. This shows that the effect of exercise decreases rapidly the longer the trial runs, possibly indicating a placebo effect that rapidly wears off. It would also be possible to include the results from the standard random effect models on the plot.

```
> exercise$duration4 <- exercise$duration - 4
> exercise$duration8 <- exercise$duration - 8
> exercise$duration12 <- exercise$duration - 12
> exercise.nodurn <- metaplus(smd, sqrt(varsmd),
+   label = "Random Mixture (No Duration)", slab = study,
+   random = "mixture", data = exercise)
> exercise.wk4 <- metaplus(smd, sqrt(varsmd),
+   mods = duration4, label = "Random Mixture (Week 4)",
+   slab = study, random = "mixture", data = exercise)
> exercise.wk8 <- metaplus(smd, sqrt(varsmd),
+   mods = duration8, label = "Random Mixture (Week 8)",
+   slab = study, random = "mixture", data = exercise)
> exercise.wk12 <- metaplus(smd, sqrt(varsmd),
+   mods = duration12, label = "Random Mixture (Week 12)",
+   slab = study, random = "mixture", data = exercise)
> plot(exercise.nodurn, extrameta = list(exercise.wk4, exercise.wk8,
+   exercise.wk12), xlab = "Effect size")
```

Conclusions and future developments

The capabilities of the **metaplus** package have been presented for fitting both standard normal random effect and robust random effect models. Using three examples it has been shown how it can test for the presence of outliers and compare the results of the robust and standard methods for both meta-analysis and meta-regression. The package has also been successfully applied to meta-analyses with larger number of studies, for example Marinho et al. (2009) with 70 studies and 3 definite outliers, and simulated data with 200 studies. One difficulty with large number of studies is the increasing computation time, especially for `testOutliers`. This will be improved by the use of parallel processing as a future enhancement.

The design of the package allows for expansion in other areas. A planned future functionality is to fit binary data, using likelihood methods based on the distribution of the binomial responses, rather than the log odds ratios fitted using a normal distribution which is the method currently used. The robust methods can then be applied in a similar way to the current models. A possible future expansion is to allow for other robust distributions although this doesn't seem necessary given the similarity of the results obtained in Baker and Jackson (2008) to those using the t-distribution.

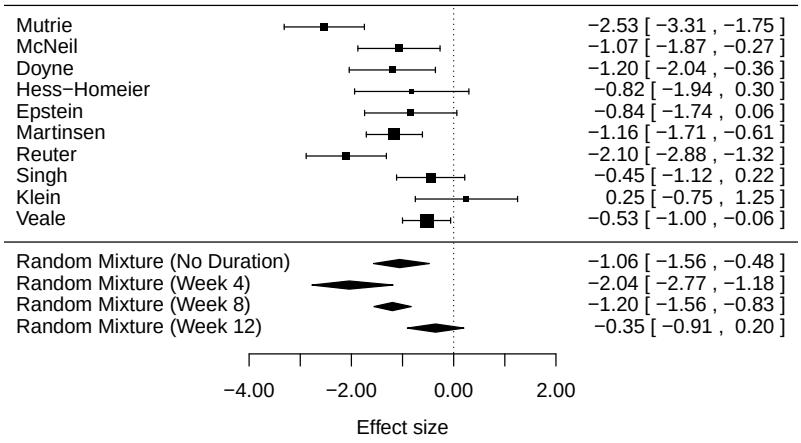


Figure 6: Forest plot for exercise versus depression studies (effect size) with summaries. Studies are sorted by increasing duration.

Acknowledgements

The author would like to thank the two anonymous reviewers for their comments and suggestions that have greatly improved the quality of the manuscript and the **metaplus** package, and provided suggestions for future enhancements.

Bibliography

A. C. Atkinson. Masking unmasked. *Biometrika*, 73(3):533–541, 1986. [p5]

R. Baker and D. Jackson. A new approach to outliers in meta-analysis. *Health Care Management Science*, 11(2):121–131, 2008. [p5, 6, 14]

K. Beath, B. Bolker, and R Core Team. *metaplus: Robust Meta-Analysis and Meta-Regression*, 2016. URL https://CRAN.R-project.org/package=metaplus. R package version 0.7-7. [p5]

K. J. Beath. A finite mixture method for outlier detection and robustness in meta-analysis. *Research Synthesis Methods*, 5:285–293, 2014. [p5, 6, 13]

B. Bolker and R Core Team. *bbmle: Tools for General Maximum Likelihood Estimation*, 2014. URL http://CRAN.R-project.org/package=bbmle. R package version 1.0.17. [p7, 9]

A. J. Branscum and T. E. Hanson. Bayesian nonparametric meta-analysis using Polya tree mixture models. *Journal of the American Statistical Association*, 64(3):825–833, 2008. [p5]

D. Burr and H. Doss. A Bayesian semiparametric model for random-effects meta-analysis. *Journal of the American Statistical Association*, 100(469):242–251, 2005. [p5]

W. Chang. *extrafont: Tools for Using Fonts*, 2014. URL https://CRAN.R-project.org/package=extrafont. R package version 0.17. [p10]

E. Demidenko. *Mixed Models: Theory and Applications with R*. Wiley, Hoboken, 2nd edition, 2013. [p5]

D. Downing. Is ISIS-4 research misconduct? *Journal of Nutritional & Environmental Medicine*, 9:5–13, 1999. [p8]

M. Fioravanti and M. Yanagi. Cytidinediphosphocholine (CDP choline) for cognitive and behavioural disturbances associated with chronic cerebral disorders in the elderly (review). *The Cochrane Database of Systematic Reviews*, 2005. URL http://onlinelibrary.wiley.com/store/10.1002/14651858.CD000269.pub2/asset/CD000269.pdf. [p11]

M. Gordon and T. Lumley. *forestplot: Advanced Forest Plot Using 'grid' Graphics*, 2015. URL https://CRAN.R-project.org/package=forestplot. R package version 1.3. [p7]

F. N. Gumedze and D. Jackson. A random effects variance shift model for detecting and accommodating outliers in meta-analysis. *BMC Medical Research Methodology*, 11, 2011. [p5, 11]

R. J. Hardy and S. G. Thompson. A likelihood approach to meta-analysis with random effects. *Statistics in Medicine*, 15:619–629, 1996. [p7]

J. P. T. Higgins and D. J. Spiegelhalter. Being sceptical about meta-analyses: A Bayesian perspective on magnesium trials in myocardial infarction. *International Journal of Epidemiology*, 31(1):96–104, 2002. [p8]

J. P. T. Higgins and S. G. Thompson. Controlling the risk of spurious findings from meta-regression. *Statistics in Medicine*, 23(11):1663–82, 2004. [p12]

ISIS-4: Collabarative Group. ISIS-4: A randomised factorial trial assessing early oral captopril, oral mononitrate, and intravenous magnesium sulphate in 58 050 patients with suspected acute myocardial infarction. *The Lancet*, 345(8951):669–682, 1995. [p8]

B. T. Johnson and T. B. Huedo-Medina. Depicting estimates using the intercept in meta-regression models: The moving constant technique. *Research Synthesis Methods*, 2(3):204–220, 2011. [p13]

E. Kontopantelis and D. Reeves. Performance of statistical methods for meta-analysis when true study effects are non-normally distributed: A simulation study. *Statistical Methods in Medical Research*, 21 (4):409–426, 2012. [p5]

J. Krogh, M. Nordentoft, J. A. C. Sterne, and D. A. Lawlor. The effect of exercise in clinically depressed adults: Systematic review and meta-analysis of randomized controlled trials. *The Journal of Clinical Psychiatry*, 72(4):529–38, 2011. [p13]

D. A. Lawlor and S. W. Hopker. The effectiveness of exercise as an intervention in the management of depression: Systematic review and meta-regression analysis of randomised controlled trials. *British Medical Journal*, 322(31 March):1–8, 2001. [p12]

K. J. Lee and S. G. Thompson. Flexible parametric models for random-effects distributions. *Statistics in Medicine*, 27:418–434, 2008. [p5, 6]

J. Li, Q. Zhang, M. Zhang, and M. Egger. Intravenous magnesium for acute myocardial infarction (review). *The Cochrane Library*, 2009. URL http://onlinelibrary.wiley.com/doi/10.1002/14651858.CD002755.pub2/pdf. [p8]

V. C. C. Marinho, J. P. T. Higgins, S. Logan, and A. Sheiham. Fluoride toothpastes for preventing dental caries in children and adolescents (review). *The Cochrane Database of Systematic Reviews*, 2009. URL http://onlinelibrary.wiley.com/doi/10.1002/14651858.CD002278/pdf. [p14]

G. J. McLachlan. On bootstrapping the likelihood ratio test statistic for the number of components in a normal mixture. *Applied Statistics*, 36(3):318–324, 1987. [p7]

G. J. McLachlan and D. Peel. *Finite Mixture Models*. Wiley, New York, 2000. [p7]

Y. Pawitan. *In All Likelihood: Statistical Modelling and Inference using Likelihood*. Clarendon Press, Oxford, 1st edition, 2001. [p7]

J. A. C. Sterne, M. J. Bradburn, and M. Egger. Meta-analysis in Stata. In M. Egger, G. D. Smith, and D. G. Altman, editors, *Systematic Reviews in Health Care: Meta-Analysis in Context*, chapter 18, pages 347–369. BMJ Publishing Group, London, 2001. [p8]

A. J. Sutton, K. R. Abrams, D. R. Jones, T. A. Sheldon, and F. Song. *Methods for Meta-Analysis in Medical Research*. Wiley, 2000. [p5, 6]

W. Viechtbauer. Conducting meta-analyses in R with the metafor package. *Journal of Statistical Software*, 36(3):1–48, 2010. [p5]

W. Viechtbauer and M. W.-L. Cheung. Outlier and influence diagnostics for meta-analysis. *Research Synthesis Methods*, 1(2):112–125, 2010. [p5]

K. L. Woods. Commentary: Biostatistics, biological mechanisms and Bayes: Lessons from the magnesium trials. *International Journal of Epidemiology*, 31:105–106, 2002. [p8]

Ken J. Beath
Department of Statistics
Faculty of Science and Engineering
Macquarie University NSW 2109
Australia
ken.beath@mq.edu.au

Gender Prediction Methods Based on First Names with genderizeR

by Kamil Wais

Abstract In recent years, there has been increased interest in methods for gender prediction based on first names that employ various open data sources. These methods have applications from bibliometric studies to customizing commercial offers for web users. Analysis of gender disparities in science based on such methods are published in the most prestigious journals, although they could be improved by choosing the most suited prediction method with optimal parameters and performing validation studies using the best data source for a given purpose. There is also a need to monitor and report how well a given prediction method works in comparison to others. In this paper, the author recommends a set of tools (including one dedicated to gender prediction, the R package called **genderizeR**), data sources (including the genderize.io API), and metrics that could be fully reproduced and tested in order to choose the optimal approach suitable for different gender analyses.

Introduction

The purpose of the **genderizeR** package and this paper is to provide tools and methods for accurate classification of various types of character strings into gender categories. An increased number of studies require gender identification, as for example, biographical research, when we want to know what is the gender of article authors and we do not have explicit gender data (Larivière et al., 2013b; West et al., 2013; Blevins and Mullen, 2015). Predicting gender of customers for marketing purposes can serve as an example from outside the academia. The **genderizeR** package makes it possible to predict gender related to a character string without knowing which term in the string is in fact a given name. Moreover, the package provides convenient built-in tools for assessing different kinds of error rates specific to gender prediction.

One of the purposes of this paper is to argue that while using informal, crowd-sourced and not widely recognized data sources, one can achieve high gender prediction efficiency comparable to other recognized gender data sources. Moreover, this effect can be obtained with less efforts and higher automatization. The paper explains how we can train models for gender prediction and how we can evaluate those models. There is also a comparison and evaluation of different approaches to gender predictions from other studies.

There have been several approaches proposed for gender prediction based on first names. Some of these methods were used in bibliometrics studies that were published in prestigious scientific journals: Larivière et al. (2013b) or West et al. (2013). One of the goals of this study is to compare the efficiency and accuracy of gender prediction methods used in the mentioned studies and consider a new approach proposed in this paper, which is easier in implementation and usage and yields outcomes comparable or, in some situations, even better than other methods.

For the purpose of method comparison, two methods were chosen from the studies: *The Role of Gender in Scholarly Authorship* (West et al., 2013) and the *Supplementary Information to Global Gender Disparities in Science* (Larivière et al., 2013a), which is the methodological appendix to *Bibliometrics: Global Gender Disparities in Science* (Larivière et al., 2013b). In these studies, authors were predicting and analysing the gender of *authorships*. The instance of an authorship had been defined as *a person and a paper for which the person is designated as a co-author* (West et al., 2013, p. 3) or even simpler as *unique paper-author combination* (Larivière et al., 2013a, p. 6). The sample of authorships also served as a common dataset for comparison of the gender prediction methods (see Section *The comparison of methods*).

Another goal of this study is to find and implement as an R package the most effective gender prediction method that is based on first names and has the following qualities:

- is based on data sources that are available for anyone in a machine-readable format,
- is fully reproducible at any given time,
- is resource effective,
- can be used to update and improve gender predictions over time with new first name data,
- is applicable for multinational studies in various research contexts,
- outperforms other similar methods in terms of accuracy of gender prediction.

Gender prediction accuracy can be defined in different ways, which implies that different metrics can be used to measure it. Later, the author will present a set of metrics that can be used in validation and comparative studies.

In both previously mentioned studies, the authors used open data with information on first names with probable corresponding gender. In order to compare the effectiveness of methods and data sources, we need to be able to reproduce those methods and reuse the same datasets. This is possible, at least to some extent, although there are several issues with the reproducibility of the studies that will be addressed in the following sections.

Review of methods and open data sources

US Census and other data sources

In the first analysed study (Larivière et al., 2013a) authors used both universal and country-specific first name lists. The sources of these data were the *US Census*, *Quebec Census*, *WikiName* portal, multiple *Wikipedia* pages, *top-100-baby-names-search.com* portal, and a few other webpages (Larivière et al., 2013a). In case of some languages, a rule-based approach was used to assign gender based on the *suffix* of a first name. In addition, human coders were used in the study. In the case of 12,828 Chinese names of authors (15.17% of total) with at least 20 papers, the gender was assigned manually by two native speakers from China. They coded the gender of each Chinese name based on their individual knowledge of the Chinese language (Larivière et al., 2013a, p. 4).

The US Census was the primary source of data for gender prediction in the study. In cases where the first name was used for both genders, it was only attributed to a specific gender when it was used at least ten times more frequently for one gender than for the other (Larivière et al., 2013a, p. 2). This rule can be converted to a probability threshold that equals **0.91** or more for the purpose of comparative analysis in this study.

With the methods applied, the paper's authors were able to predict gender for **86%** out of 21 million authorships with full first names from the *Web of Science* database (Larivière et al., 2013a); nevertheless, several major drawbacks of this approach can be identified:

- The presented analysis is **difficult to reproduce**. The full set of first names with corresponding gender data used in this study is not available in machine-readable format, and some data sources even ceased to exist. For example, the *wiki.name.com* portal is not accessible any more (assessed on January 16, 2015).
- The manual coding of gender by humans is **neither fast nor cost-effective**. Moreover, it could be **not reliable enough**, as in the paper there was no information about manual coding accuracy and inter-coder reliability.
- The sources of the data are **not in easily readable machine formats** with the significant exception of the data from the US Census, which can be obtained as text files (United States Census Bureau, 2015a) or as a data directly from the R package **qdap** (Rinker, 2013). In order to use other gender data, they need to be web-scraped and parsed. Moreover, not all *Wikipedia* webpages with country-specific first names have a standardised HTML structure that can be easily utilised in parsing HTML code.
- In this mixed data source approach the **confidence threshold cannot be easily changed**, and researchers are only able to use predefined name categories, such as male, female and unisex. The **category unisex is not very informative**, as it means that we predict that the gender can be either female or male with an unknown probability. Moreover, such a category can be easily recreated when the probability of being a male or a female is known, given the first name (e.g., we can set the predicted category as unisex for all first names that have a 0.5 probability of being male names). In a case when predicted gender is unisex without additional information, it is impossible to precisely assess the effectiveness of such a gender prediction method.

On the other hand, the main advantages of the described approach are as follows:

- Usage of different techniques and country-specific sources in gender prediction could increase the percentage of different types of items with correctly predicted gender and, above all, **decrease the percentage of items with unpredicted gender**.
- There is a strong possibility that at least some of the open data sources used in the analysis will be updated over time, for example *Wikipedia* webpages, and will **include new first names or infrequently used names** in the future. This is not certain, however, as even the US Census Bureau has not provided a newer first name dataset since the 2000 Census (United States Census Bureau, 2015b).

The article by Larivière et al. (2013a) does not explicitly mention **any recognised prediction accuracy metrics** that can be comparable with other gender prediction methods, although its confusion matrix can be rebuilt from the tables with the data from the validation study (see Section *The comparison of methods*).

US Social Security Administration records

In the second analysed approach (West et al., 2013) authors used *US Social Security Administration* (SSA) records with the **top 1000** first names annually collected for each of the 153 million boys and 143 million girls born in the USA from 1880–2010. The authors also have decided *a priori* to use only those records that have at least a **95%** probability to correctly predict gender based on a first name (West et al., 2013).

Based on this method, the authors were able to predict gender for **73%** out of 3 million authorships with full first names from the JSTOR network dataset (Efron, 1983, pp. 2–3).

The main drawbacks of this approach are:

- Non-US first names and names that do not appear in the top 1000 first names cannot be used for gender prediction, so **the first name dataset is US-specific** and is not comprehensive by its definition.
- The authors of the analysis **utilised limited information of the top 1000 baby first names**, although SSA provided an extended version of this database with baby names that occur at least five times in the years between 1880–2013. That extended dataset has information on about 92 600 unique baby names compared to 6 983 unique baby names in the top 1000 dataset.

The main advantages of this approach are:

- The US SSA **baby first names database is updated every year** and is available for anyone as open data in machine, easy-readable format (Social Security Administration, 2015).
- The method is **fully and easily reproducible**, especially with the use of R packages like **gender** (Mullen, 2014) or **babynames** (Wickham, 2014) where the full baby name data provided by the SSA is included as built-in datasets.

The paper does not report **any gender prediction accuracy metrics** (West et al., 2013).

Social network profiles as gender data source (via the genderize.io API)

The third tested approach is our proposition of gender prediction based on first name and gender data from the *genderize.io* database, which was created by Casper Strømgren (Strømgren, 2015a) in August 2013 and has been regularly updated since. Regular incremental updates are possible due to continuous scanning of public profiles and their gender data in major social networks. The database is continuously growing by processing approximately **from 15 000 to 20 000 social network profiles per day**.

On 24-th of May 2014, the *genderize.io* database contained information on **120 517** terms that at least once had been used as a first name in about half a million social network profiles. In April 2015 there were **212 252** unique terms gathered from about **2 million** social network profiles from **79 countries** in **89 languages** (Strømgren, 2015a).

A quick connection to the *genderize.io* database is possible through its *application programming interface* (API). Since February 2014, database queries via the *genderize.io* API have been restricted to 10 terms per request and 1000 terms per day to prevent server overload. Higher limits are easily available through commercial access plans to the API and enable checking up to 10 000 000 names monthly (Strømgren, 2015b).

As a query term to the database, any character string can be used if it is suspected to be a first name (a simple example of a query: `GET http://api.genderize.io?name=peter`). In response to the query, the API returns a null value when the string is not found in the *genderize.io* database. If the term is found in the database, the API returns several values in *JavaScript Object Notation* (JSON) format: a predicted gender for the first name (male or female) and two numeric values that can be further use to customise the gender prediction. These values are *count* and *probability*, where *count* shows how many social network profiles in the database have been recorded with this particular term as a first name and *probability* shows the proportion of profiles with the first name and the predicted gender (a simple example of API response: `"name":"peter","gender":"male","probability":"1.00","count":4300`) (Strømgren, 2015a). Therefore, we not only know the gender prediction for a given term but also how confident we can be with this particular prediction.

Using the *genderizer.io* database through its API for predicting gender has strong advantages:

- The *genderizer.io* database is **continuously and incrementally growing**, thus we are not only able to reproduce classification results using previously saved API output, but we also could improve our predictions next time using newer API output from a larger, updated, and more comprehensive version of the first name database.

Characteristics of approach	Lariviere et al. 2013b	West et al. 2013	*genderize.io*
main data source	US Census	US SSA	public social profiles
other data sources	yes	no	no
open data	some datasets	yes	limited free access
machine-readable format	some datasets	yes	yes
API connection	no	no	yes
easily reproducible	no	yes	yes
resource effective	no	yes	yes
regular data updates	no	yearly	in real time
known probabilities of gender prediction	available only in the main data source	available	available
global reach	country-specific	country-specific	yes

Table 1: Comparison of the characteristics of different approaches to gender prediction based on first names.

- The communication with the API is **fast, effective, and straightforward**; additionally comprehensive documentation is available (Strømgren, 2015a). Communication through the API can be further simplified with the use of dedicated functions in the **genderizeR** package (Wais, 2016).

The issue that can be clearly seen as disadvantage is the daily limit of free queries through the API (1000 terms per day). Much larger limits are still available through commercial plans with reasonable prices. This kind of commercial model behind the *genderize.io* API has also some advantages in comparison with completely free access to the API. It guarantees stability of the service and constant development of the database, covers costs of the servers, and prevents server overloads due to unrestricted access.

The main criticism of this data source is the reliability of the data. The database behind the *genderize.io* API draws on data from numerous public social media profiles, although neither the exact number of sources nor the total number of profiles have been revealed. Even if a social media portal has a real-name policy implemented, there is no guarantee that at a given time each profile has valid and reliable data in its first name and gender fields. So reliability of the data from a perspective of a single profile is very low. However, it is safe to assume that most people give true information regarding their gender and given name, thus such crowd-sourced data aggregated from many profiles can give reliable information due to the scale of the constantly growing database. The major drawback of this data source is the noise in the data related to their declarative character. While creating a profile, one can submit any character string in a given names field. Even if the user profile is corrected later, that bogus data could already be recorded in the *genderize.io* database. This is the obvious disadvantage compared to other official gender data sources, although the noise can be reduced by setting a higher threshold for profile counts. In this way, we are able to use only those terms for given names and gender data that were also entered in some other social media profiles and therefore seem to be 'confirmed' by other users.

Comparison of approach characteristics

Table 1 shows that the Larivière et al. (2013a) approach is based on a resource-consuming method and lacks some important characteristics like reproducibility. The approach in West et al. (2013) is fully reproducible and enhanced with other important characteristics, but the data source used is US-specific and infrequently updated. Utilising the *genderize.io* database via its fast API, we gain access to a global database of first names that is being continuously updated even at this moment.

The genderizeR package

In order to provide an effective tool for performing and evaluating gender prediction methods, the **genderizeR** package for R has been built. The package enables convenient communication with the *genderize.io* API and has built-in functions for evaluating the effectiveness of gender prediction with metrics specific to this topic.

The package could be used for different tasks:

- for pre-processing text vectors for future gender prediction;
- for connecting with the *genderize.io* database through its API;
- for *genderizing* character strings, which means that the gender is predicted even if we do not indicate which term from the string is the first name. The algorithm assumes that all input terms could be potentially gender indicators and searches for the most credible one;
- for *training* gender prediction algorithms (looking for the optimal combination of *gender* and *probability* parameters from a given set in order to minimise the gender prediction accuracy metric);
- for estimation of different gender prediction accuracy metrics.

There are four main components of the package:

- functions working with the *genderize.io* API (`textPrepare`, `genderizeAPI`, `findGivenNames`);
- functions for training and predicting gender (`genderize`, `genderizeTrain`, `genderizePredict`);
- functions assessing different kinds of gender prediction errors (`classificationErrors`, `genderizeBootstrapError`);
- training datasets (`authorships`, `givenNameDB_authorships`, `titles`, `givenNamesDB_titles`).

The `genderizeBootstrapError` function is based on code from the **sortinghat** package (Ramey, 2013). The function has built-in functionality for parallel processing working directly with functions from the **genderizeR** package (`genderizeTrain` and `genderizePredict`). The parallel processing uses the **parallel** package and its implementation was inspired by Nathan VanHoudnos' scripts (`http://edustatistics.org/nathanvan/setup/mclapply.hack.R`).

The `textPrepare` function for text pre-processing helps to prepare terms for API queries. It utilizes functions from the R packages **stringr** (Wickham, 2012) and **tm** (Feinerer and Hornik, 2014; Feinerer et al., 2008) to perform a series of pre-processing steps (removing special characters, numbers and punctuation; building a vector of unique terms which can be used for creating API queries).

A trivial example of basic package functions

If we use the package for the first time, we need to install it from the *Comprehensive R Archive Network* (CRAN) or from the *GitHub* repository. The last stable version of the package is on CRAN, and the latest development version of the package is available from the GitHub repository "kalimu/genderizeR" (Wais, 2016).

With the `findGivenNames` function we can easily look for first names in the *genderize.io* database. In return, we obtain a data table with the following records: the terms that appear in the database as first names, their predicted gender, probability of such a prediction, and the count of profiles on which the prediction is based . The outcome is alphabetically sorted by the *name* column, and all terms that appear to be first names are lower-cased. Because the outcome is generated from the current state of the *genderize.io* database, to reuse exactly the same outcome later, we should save it locally. This is an important step in reproducible analysis because if we run the same function next time, our output could be based on a larger count of social network profiles and could give us a slightly different data. The `progress = FALSE` argument turns off the progress bar.

```
R> library('genderizeR')
R> findGivenNames(c('Marie', 'Albert', 'Iza', 'Olesia', 'Marcin', 'Andrzej', 'Kamil'),
+                 progress = FALSE)

      name gender probability count
1:  albert   male        0.99   710
2: andrzej   male        0.98    49
3:     iza female        1.00    28
4:   kamil   male        0.99   124
5:  marcin   male        1.00   128
6:   marie female        0.99  2248
7:  olesia female        1.00     4
```

In some cases, we might need to predict the gender of a person based on the full name. It is trivial when we know which term is a first name (or first names) because we can manually extract it and check its gender with the `findGivenNames` function. When we do not know exactly which term in a character vector is a first name, the same function attempts to predict gender by analysing unique terms in the vector.

This way we could predict the gender of an author of an article without explicitly extracting the first name from the full name. For example, one biographical article from the *Web of Science* (WOS)

database records is titled "Marie Sklodowska-Curie (1867–1934)". We know that the author's full name of the article is recorded as "Pascual-Leone Pascual, Ana Ma" (WOS, 2014).

```
R> x <- 'Pascual-Leone Pascual, Ana Ma'
```

We can use the `findGivenNames` function directly on our vector. The function first calls another function from the package, `textPrepare`, which builds a vector of unique terms that are used in queries to *genderize.io* API. This is more effective than checking the same term many times through the API. In the outcome of the `textPrepare` function, we have terms which are at least two characters long, as we can assume that a one-character term could not be a full first name. The text pre-processed tasks that `textPrepare` function can perform are:

- removing single-character terms (like initials),
- removing punctuation,
- removing special characters (like exclamation marks, question marks, hyphens, etc.),
- removing numbers,
- converting all characters to lowercase,
- striping white-spaces.

```
R> textPrepare(x)
[1] "ana"     "leone"   "ma"      "pascual"
```

In the next step, we can check our terms in the *genderize.io* database and store the output for later use.

```
R> (genderDB <- findGivenNames(x, progress = FALSE))

      name gender probability count
1:     ana female        0.99  3535
2:   leone female        0.81    27
3:      ma female        0.62   243
4: pascual   male        1.00    25
```

All four unique terms occur in the *genderize.io* database and could be used as gender indicators, but the *count* value suggests which terms are indeed true first names in our example. The term 'Ana' has the largest *count* value, so it will be used to predict gender for the given author with the `genderize` function.

```
R> genderize(x, genderDB = genderDB, progress = FALSE)

                          text givenName gender genderIndicators
1: Pascual-Leone Pascual, Ana Ma       ana female                4
```

In the final step, we used the previous output as gender information for our terms. We use an output from the `findGivenNames` function, but it could also be a dataset from the US Census or US SSA, although it should be converted to the same format with the same column names.

The `genderize` function output contains our original author's full name in the `text` column, the term that was assumed to be the gender indicative first name (`givenName` column), and the predicted gender (`gender` column). There are also `genderIndicators` values that show how many terms in a `text` were found in our gender dataset. The `givenName` term is the one that won the competition between terms to be a final gender indicator. To find the winning term, we compare the counts for all terms found and choose the one with the maximum value. It is a reasonable way to prevent prediction based on accidental terms.

It should be noted that in cases when a person has a double first name like 'Hans-Peter' the terms 'Hans' and 'Peter' are looked up in the database separately and the gender prediction is still done based on the term with more counts. This is a counter-intuitive solution as the *genderize.io* API can accept queries with double first names. However it simplifies the algorithm and still gives proper predictions even if the double name is not present in the database.

```
R> x <- 'Hans-Peter'
R> (genderDB <- findGivenNames(x, progress = FALSE))

    name gender probability count
1:  hans   male        0.99   425
2: peter   male        1.00  4300
```

```
R> genderize(x, genderDB = genderDB, progress = FALSE)

         text givenName gender genderIndicators
1: Hans-Peter     peter   male                2
```

In the same way, we can predict gender not only from authors' full names but also from larger character strings, for example, titles of the biographical articles. In the next example, we have six such titles in which some person names are mentioned. Half of the articles in this example are about Marie Sklodowska-Curie and half are about Albert Einstein. Let us assume that we do not know which title concerns a female and which a male, and we want to check it automatically.

```
R> x <- c('Marie Sklodowska-Curie (1867-1934) - A person and scientist',
+         'Albert Einstein as a philosopher of science',
+         'The legacy of Albert Einstein (1879-1955)',
+         'Life and work of Marie Sklodowska-Curie and her family',
+         'Marie Sklodowska-Curie (1867-1934)',
+         'Albert Einstein, radical - A political profile')
R> (givenNamesDB <- findGivenNames(x, progress = FALSE))

        name gender probability count
1:    albert   male        0.99   710
2:        as   male        0.89    64
3:  einstein   male        1.00     4
4:    family   male        1.00     1
5:       her female        0.50     8
6:    legacy   male        1.00     2
7:     marie female        0.99  2248
8: political   male        1.00     1
9:       the female        0.50     2
```

We should note that cases are possible where the number of females is the same as the number of males (the probability equals 0.50, and the count is an even number). In our example, such a situation occurs with the *her* term. In such situations, the API returns female as gender prediction, but in the future it will likely be changed to the more valid unisex category.

There are some noise in the gender data obtained from the *genderize.io* API so we should approach the results from the `findGivenNames` function critically. In the above example the terms "as", "the" and "her" seem to be used in some profiles even if they do not stand for a given name. We can deal with such terms by setting the `count` threshold high enough (ex. `count >= 100`) or by using a list of non-acceptable words (*blacklist / stopwords*). The second solution can be implemented to the result of the `textPrepare` and before using `findGivenNames` function. This way we will also reduce the usage of API requests. Using a counts threshold we need to remember that its height should be relative to the growing number of profile data processed by *genderize.io*. However, in our example the noise does not impact the accuracy of the gender predicion.

```
R> genderize(x, genderDB = givenNamesDB, progress = FALSE)

                                                          text
1: Marie Sklodowska-Curie (1867-1934) - A person and scientist
2:                 Albert Einstein as a philosopher of science
3:                   The legacy of Albert Einstein (1879-1955)
4:      Life and work of Marie Sklodowska-Curie and her family
5:                           Marie Sklodowska-Curie (1867-1934)
6:              Albert Einstein, radical - A political profile
   givenName gender genderIndicators
1:     marie female                1
2:    albert   male                3
3:    albert   male                4
4:     marie female                3
5:     marie female                1
6:    albert   male                3
```

We can also set the thresholds for *probability* or *count* values to exclude accidental terms or unisex first names from being considered in a gender prediction. This method can be used to train the prediction algorithm in order to minimise the classification error. However, we need to be careful while setting the thresholds because, if we set them too high, we could unintentionally exclude some true first names and the algorithm will not be able to predict gender, returning only `NA` values.

```
R> genderize(x, givenNamesDB[count > 2000, ], progress = FALSE)

                                                              text
1: Marie Sklodowska-Curie (1867-1934) - A person and scientist
2:                 Albert Einstein as a philosopher of science
3:                   The legacy of Albert Einstein (1879-1955)
4:      Life and work of Marie Sklodowska-Curie and her family
5:                           Marie Sklodowska-Curie (1867-1934)
6:              Albert Einstein, radical - A political profile
   givenName gender genderIndicators
1:     marie female                1
2:        NA     NA                0
3:        NA     NA                0
4:     marie female                1
5:     marie female                1
6:        NA     NA                0
```

The communication with the *genderize.io* API is based on UTF-8 encoding. We can also make use of a specific locale.

```
R> Sys.setlocale("LC_ALL", "Polish")
R> (x <- "Róża")
[1] "Róża"

R> (xp <- textPrepare(x))
[1] "róża"

R> findGivenNames(xp, progress = FALSE)
#    name gender probability count
# 1: róża female        0.89    28
```

If the `findGivenNames` function stops due to the API free or commercial plan limits the results from already performed queries are returned and can be saved as an object.

```
R> xPrepared <- textPrepare(authorships[['value']][1:1200])
R> givenNames_part1 <- findGivenNames(xPrepared)

Terms checked: 10/86. First names found: 4.          |   0%
Terms checked: 20/86. First names found: 7.          |  11%
Terms checked: 30/86. First names found: 12.         |  22%
Terms checked: 40/86. First names found: 17.         |  33%
Terms checked: 50/86. First names found: 22.         |  44%
Terms checked: 60/86. First names found: 25.         |  56%
  |================================                  |  67%
 Client error: (429) Too Many Requests (RFC 6585)
 Request limit reached

The API queries stopped at 57 term.

Warning messages:
1: In genderizeAPI(termsQuery, apikey = apikey, ssl.verifypeer = ssl.verifypeer) :
   You have used all available requests in this subscription plan.
2: In findGivenNames(xPrepared) : The API queries stopped.
```

Moreover the function reports an index of the last term which has been successfully queried before reaching the API limit. This enables us to start the API queries from that term when the API plan resets the next day.

```
R> givenNames_part2 <- findGivenNames(xPrepared[57:NROW(xPrepared)])
```

Finally, we can bind all parts together.

```
R> givenNames <- rbind(givenNames_part1, givenNames_part2)
```

Sample datasets

In the **genderizeR** package we have two sample datasets: `authorships` and `titles`. Both datasets contain data from a simple random sample of biographical articles from the WOS database records of articles of *biographical-items* or *items-about-individual* type from all fields of study, published from 1945 to 2014. The sample was drawn in December 2014. The first dataset `authorships` contains 2 641 authorships of 2 000 randomly sampled records of biographical articles.

Part of the authors in this dataset were successfully identified and manually coded as females or males (`genderCoded` column). Human coders based their coding decisions on results from Internet queries regarding full names of the authors and their affiliations, biographies, mentions in the press and photos. If a full name of an author was unavailable the gender was coded as `noname`; if a full name was available but the coder was not able to code the author's gender with a high degree of certainty — the record was coded as `unknown`.

```
R> tail(authorships[, c(4, 5)])
```

```
             value genderCoded
2636  Morison, Ian        male
2637 Hughes, David        male
2638 Higson, Roger        male
2639    CONDON, HA      noname
2640  GILCHRIST, E      noname
2641     Haury, LR      noname
```

The second dataset `titles` contains 1 190 titles of biographical articles, which also were manually coded as female or male.

```
R> tail(titles)[-3, ]
```

```
                                          title genderCoded
1:   Yuri A. Chizmadzhev (to the 80th anniversary)        male
2:           Yurko Duda, a physicist like few ones        male
3: Zhores Ivanovich Alferov (on his 80th birthday)        male
4:          Zongluo Luo, a Chinese Haigui in 1930s        male
5:                                    lynn seymour      female
```

These datasets can be used to assess the efficiency and accuracy of gender prediction for different prediction methods and related data sources.

For reproducibility purposes, in the package there are two additional datasets corresponding to the previous two: `givenNamesDB_authorships` and `givenNamesDB_titles`. Both are outputs of the `findGivenNames` function, which was used on authorship and title vectors on December 26, 2014. These datasets can be used for gender prediction without the necessity of connecting to the *genderize.io* database, and to provide reproducible outcomes. Using API queries is also possible, although it will probably give slightly different outputs due to the first name data updates in the database.

Selecting prediction parameters

Knowing that each first name record from the *genderize.io* database has been assigned a probability of predicted gender, we can utilise these pieces of information to strengthen the confidence of gender prediction. Moreover, for each record, the number of instances of social network profiles that were used to calculate such a probability is known. Thus, we can set our own thresholds of *counts* and *probabilities* when we need it, and choose settings that can optimise chosen prediction efficiency metrics.

Metrics of gender prediction efficiency

To assess the efficiency of gender prediction, we need to agree on the set of efficiency indicators that are adequate to this particular classification problem. We can calculate some of the indicators with the help of `classificationErrors` function that are based on a prediction confusion matrix – as an example below of randomly generated samples of labels and predictions.

```
R> set.seed(238)
R> labels <- sample(c('male', 'female', 'unknown'), size = 100, replace = TRUE)
R> predictions <- sample(c('male', 'female', NA), size = 100, replace = TRUE)
```

```
R> indicators <- classificationErrors(labels, predictions)
R> indicators[['confMatrix']] # confusion matrix for the generated sample

          predictions
labels     female male <NA>
  female       12   10    4
  male          7   10   12
  unknown      16   13   16
  <NA>          0    0    0
```

For the confusion matrix, we could calculate a standard classification error rate (one minus sum of the diagonal divided by sum of total), but this is not directly applicable in this specific classification problem; some titles were manually coded as `unknown` gender if they do not contain a first name or when we could not verify the gender of the person mentioned in a title. In this case, we should not blame the algorithm for wrongly predicting gender, considering the human coder could not do it either. Instead, we can calculate the **coded error rate** on the confusion matrix with manual female and male codes only.

```
R> # errorCoded <- (7 + 10 + 4 + 12) / (12 + 10 + 4 + 7 + 10 + 12)
R> unlist(indicators['errorCoded'])

errorCoded
       0.6
```

The advantage of such an error rate calculation is that it incorporates within itself all items with unpredicted gender (`<NA>`). Therefore, if the algorithm is unable to predict gender, we treat it as a prediction error and thus penalise our prediction efficiency metric.

Another possibility is to calculate the coded error rate but only for a matrix with automatically predicted gender (without any `NA` values).

```
R> # errorCodedWithoutNA <- (7 + 10) / ( 12 + 10 + 7 + 10)
R> unlist(indicators['errorCodedWithoutNA'])

errorCodedWithoutNA
          0.4358974
```

Such a **coded error rate without `NA` values** indicates how well the algorithm predicts gender, provided that prediction is possible. With a high threshold of gender probability, we increase the prediction accuracy, but we also risk that for many first names the algorithm will not be able to predict their gender. Therefore, the cost of high prediction accuracy can be measured as the increase of the proportion of items with unpredicted gender.

```
R> # naCoded <- (4 + 12) / ( 12 + 10 + 4 + 7 + 10 + 12)
R> unlist(indicators['naCoded'])

  naCoded
0.2909091
```

There is another problem-specific error that we can calculate from the confusion matrix. It can be called the **gender bias error rate**, and can be calculated as a difference between the number of items manually labelled as female but automatically classified as male and the number of items manually classified as male but automatically classified as female, and all that are divided by the number of all items labelled or classified as female or male.

```
R> # errorGenderBias <- (7 - 10) / (12 + 10 + 7 + 10)
R> unlist(indicators['errorGenderBias'])

errorGenderBias
    -0.07692308
```

We can interpret the sign of such an indicator as a direction of the bias in gender prediction. Negative values suggest that more females are incorrectly classified as males than males as females, hence the predicted proportion of females could be slightly overestimated. If the value is positive, the proportion of females are underestimated. When the bias is close to 0, we have the case where nearly the same numbers of both female and male items are classified wrongly as males and females, respectively. Therefore, in the trivial example, if we have a sample of 10 males and 10 females and we

Prediction indicator	What items are taken into account?
`errorCoded`: Coded Error Rate	Items with manually coded female and male labels; items with predicted female and male labels, or with unpredicted gender.
`naCoded`: Proportion of Items with Unpredicted Gender	Items with manually coded female and male labels; items with predicted female and male labels, or with unpredicted gender.
`errorCodedWithoutNA`: Coded Error Rate without `NA` Values	Items with manually coded female and male labels, and with predicted female and male labels only.
`errorGenderBias`: Gender Bias Error Rate	Items with manually coded female and male labels, and with predicted female and male labels only.

Table 2: Comparison of selected gender prediction indicators.

incorrectly classify five males as females and five females as males, we will still have a proportion of 50% of female items in the sample, and the bias error is 0.

A brief comparison of the described indicators is shown in Table 2. The table indicates that we always focus on metrics based on manually coded gender labels with the exception of two indicators, where we consider items with unpredicted gender as well.

Depending on the research goals, we can utilise one or more of these metrics, for example:

- in a scenario where the goal is to predict gender for as many items as possible, we can try to minimise the **coded error rate**,
- in a scenario where the prediction accuracy is more important than the percentage of items with unpredicted gender, we can try to minimise the **coded error rate without `NA` values**,
- in a scenario where we primarily want to accurately estimate the proportion of males or females, we can try to minimise the **gender bias error rate**.

Different decisions can be made corresponding to different research needs. We can try to minimise all these indicators at the same time and look for their optimal values based on our research questions.

Case study 1: Authorships

In order to establish optimal values of gender prediction indicators for the authorship sample dataset, we can manipulate the thresholds of `probability` and `count` values. In order to do so, we can use the `genderizeTrain` function with values that we prefer to be considered in the *training* of the algorithm as the function arguments `probs` and `counts`. In this case, we can utilise some typical probability values used for gender prediction together with different values of counts that will give noticeable patterns of error rates for parameter combinations.

```
R> probs <- c(0.5, 0.7, 0.8, 0.9, 0.95, 0.97, 0.98, 0.99, 1)
R> counts <- c(1, 10, 100)
R> authorshipsGrid <-
+    genderizeTrain(# parallel = TRUE,
+                   x = authorships$value,
+                   y = authorships$genderCoded,
+                   givenNamesDB = givenNamesDB_authorships,
+                   probs = probs,
+                   counts = counts)
```

The `genderizeTrain` function first creates the grid of parameters for all unique combinations of `probs` and `counts` values. Then, prediction is done based on `givenNamesDB` data trimmed by `probs` or `counts` parameters. As the output, we attain gender prediction defectiveness indicators for all combinations of parameters (27 in our example below).

```
R> authorshipsGrid
```

```
   prob count errorCoded errorCodedWithoutNA    naCoded errorGenderBias
 1: 0.50     1 0.07093822          0.03791469 0.03432494     0.014218009
 2: 0.70     1 0.08466819          0.03147700 0.05491991     0.007263923
 3: 0.80     1 0.10983982          0.03233831 0.08009153     0.012437811
 4: 0.90     1 0.11899314          0.03022670 0.09153318     0.015113350
 5: 0.95     1 0.13272311          0.02820513 0.10755149     0.012820513
 6: 0.97     1 0.14645309          0.02610966 0.12356979     0.010443864
 7: 0.98     1 0.15560641          0.02638522 0.13272311     0.010554090
 8: 0.99     1 0.18306636          0.02724796 0.16018307     0.005449591
 9: 1.00     1 0.27459954          0.03353659 0.24942792    -0.003048780
10: 0.50    10 0.12128146          0.03759398 0.08695652     0.017543860
11: 0.70    10 0.13958810          0.02842377 0.11441648     0.007751938
12: 0.80    10 0.16247140          0.02917772 0.13729977     0.013262599
13: 0.90    10 0.16933638          0.02680965 0.14645309     0.016085791
14: 0.95    10 0.18535469          0.02465753 0.16475973     0.013698630
15: 0.97    10 0.19908467          0.02234637 0.18077803     0.011173184
16: 0.98    10 0.20823799          0.02259887 0.18993135     0.011299435
17: 0.99    10 0.23569794          0.02339181 0.21739130     0.005847953
18: 1.00    10 0.33180778          0.02666667 0.31350114     0.000000000
19: 0.50   100 0.27459954          0.03058104 0.25171625     0.012232416
20: 0.70   100 0.29061785          0.02821317 0.27002288     0.009404389
21: 0.80   100 0.30892449          0.02893891 0.28832952     0.016077170
22: 0.90   100 0.31350114          0.02912621 0.29290618     0.016181230
23: 0.95   100 0.32036613          0.02941176 0.29977117     0.016339869
24: 0.97   100 0.32951945          0.02657807 0.31121281     0.013289037
25: 0.98   100 0.33638444          0.02684564 0.31807780     0.013422819
26: 0.99   100 0.36613272          0.02807018 0.34782609     0.007017544
27: 1.00   100 0.45995423          0.02880658 0.44393593     0.004115226
```

We could also visualise the prediction effectiveness of the parameter grid on a scatterplot and look for optimal values of indicators (Figure 1). In this case, we will use only a subset of probability values (0.5, 0.8, 0.95, 0.98, 1) in order to keep the scatterplot clear.

If the goal is to minimise the coded error rate, we should set the threshold values as low as possible (`prob = 0.5` and `count = 1`).

```
R> authorshipsGrid[authorshipsGrid$errorCoded == min(authorshipsGrid$errorCoded), ]

  prob count errorCoded errorCodedWithoutNA    naCoded errorGenderBias
1:  0.5     1 0.07093822          0.03791469 0.03432494      0.01421801
```

However, if the goal is to have the lowest gender bias error rate, we should set the threshold values as `prob = 0.97` and `count = 10`. However, in this way we increase our proportion of items with unpredicted gender from less then 4% to more than 18%.

```
R> authorshipsGrid[authorshipsGrid$errorGenderBias ==
+                    min(abs(authorshipsGrid$errorGenderBias)), ]

   prob count errorCoded errorCodedWithoutNA   naCoded errorGenderBias
1: 0.97    10  0.1990847          0.02234637 0.180778      0.01117318
```

Case study 2: Titles of biographical articles

As in the first case study, we can train the algorithm on the second dataset with titles of biographical articles, using a wider set of values for both `probs` and `counts` parameters. In the case of a huge parameter grid, we can also try to run the `genderizeTrain` function in parallel version (see `parallel` argument).

```
R> probs <- seq(from = 0.5, to = 0.9, by = 0.05)
R> probs <- c(probs, seq(from = 0.91, to = 1, by = 0.01))
R> counts <- seq(from = 1, to = 16, by = 1)
R> titlesGrid <-
+    genderizeTrain(# parallel = TRUE,
+                   x = titles$title,
+                   y = titles$genderCoded,
+                   givenNamesDB = givenNamesDB_titles,
```

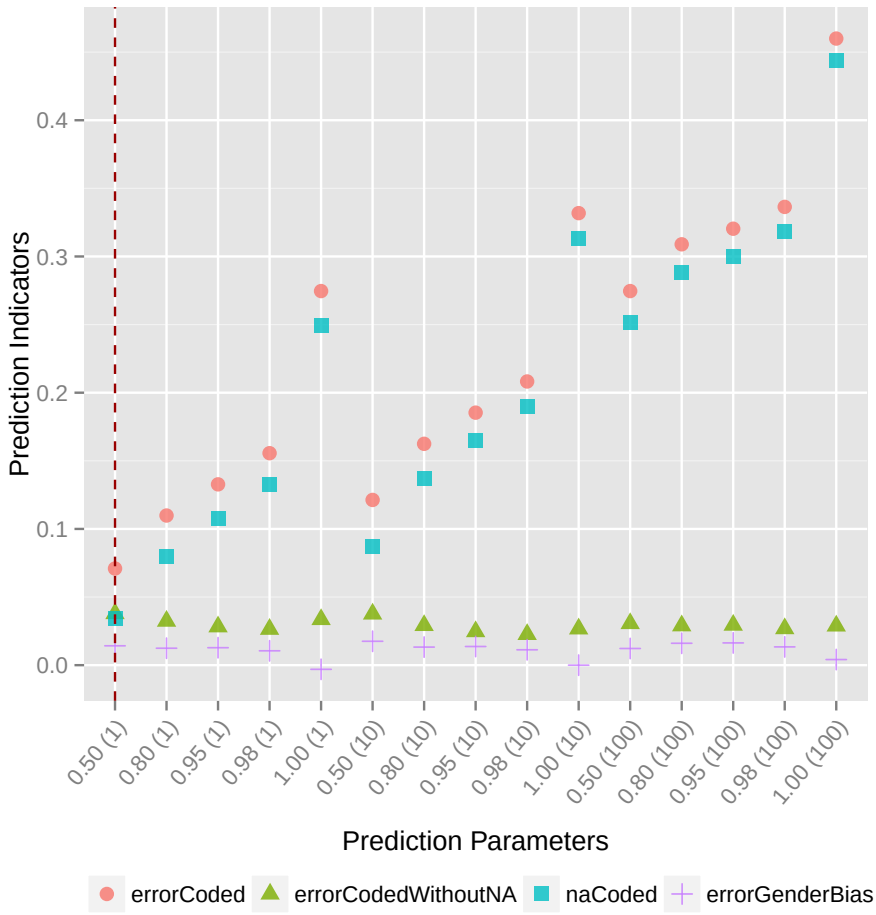


Figure 1: Effectiveness of gender prediction for given parameters from the authorship sample. The optimal effectiveness, when all indicators have the lowest values possible, seems to be achievable with probability = 0.50 and count = 1 and is marked on the plot as a dashed red line. The proportion of items with unpredicted gender (`naCoded`) increases with the increase of parameter values. The lowest gender bias error rate (`errorGenderBias`) can be achieved by setting the probability parameter to 1, although it will greatly increase the proportion of unpredicted items.

```
+                  probs = probs,
+                  counts = counts)
```

Because the scatterplot for all parameter combinations would be very unclear, we have visualised only a subset of count values: 1, 4, 7 (Figure 2). The trends and patterns that emerged from such a simplified visualisation are similar to other combinations of parameters.

Parameter values `prob = 0.70` and `count = 1` minimised coded error rate for this dataset produced a reasonable low gender bias error rate. We can also achieve lower gender bias error rate; however, it comes with an increased proportion of unpredicted items.

```
R> titlesGrid[titlesGrid$errorCoded == min(titlesGrid$errorCoded), ]

   prob count errorCoded errorCodedWithoutNA    naCoded errorGenderBias
1:  0.7     1  0.1004367          0.06533575 0.03755459      0.02540835
R> titlesGrid[titlesGrid$errorGenderBias ==
+             min(abs(titlesGrid$errorGenderBias)), ]

   prob count errorCoded errorCodedWithoutNA   naCoded errorGenderBias
1:    1    15  0.2917031          0.04023669 0.2620087     0.002366864
2:    1    16  0.2917031          0.04023669 0.2620087     0.002366864
```

The final decision on which set of parameter values to use can be subjective and is dependent on the importance of each indicator in a given analysis.

Further estimation of prediction accuracy

For further estimation of gender prediction accuracy we can use cross-validation or bootstrap methods like *Leave-One-Out Bootstrap* (LOO) (Efron, 1983) and *.632+ Rule* (Efron and Tibshirani, 1997) as well as *Receiver Operating Characteristic* (ROC) and *Area Under Curve* (AUC) (Fawcett, 2003) or finally *Brier Score* (Brier, 1950) as one of the most popular performance metrics for probabilistic classifiers.

Bootstrapping

Up to this moment, we have only relied on the so called *apparent error rate*, which is *the observed inaccuracy of the fitted model to the original data points*. However, the apparent error rate usually underestimates the true error rate and gives a falsely optimistic picture of the model's true accuracy (Efron, 1986). To address this issue, we can calculate the *Leave-One-Out (LOO) bootstrap error rate* proposed by Efron (1983).

In the LOO bootstrap procedure, we generate some number of bootstrap samples of the size of the original sample. On each bootstrap sample, we can train our prediction algorithm by minimising the classification error. Later, we can use optimal parameters to predict gender for items that have not been sampled in this particular bootstrap sample. This way, we avoid testing a prediction model on the items used for choosing the prediction parameters.

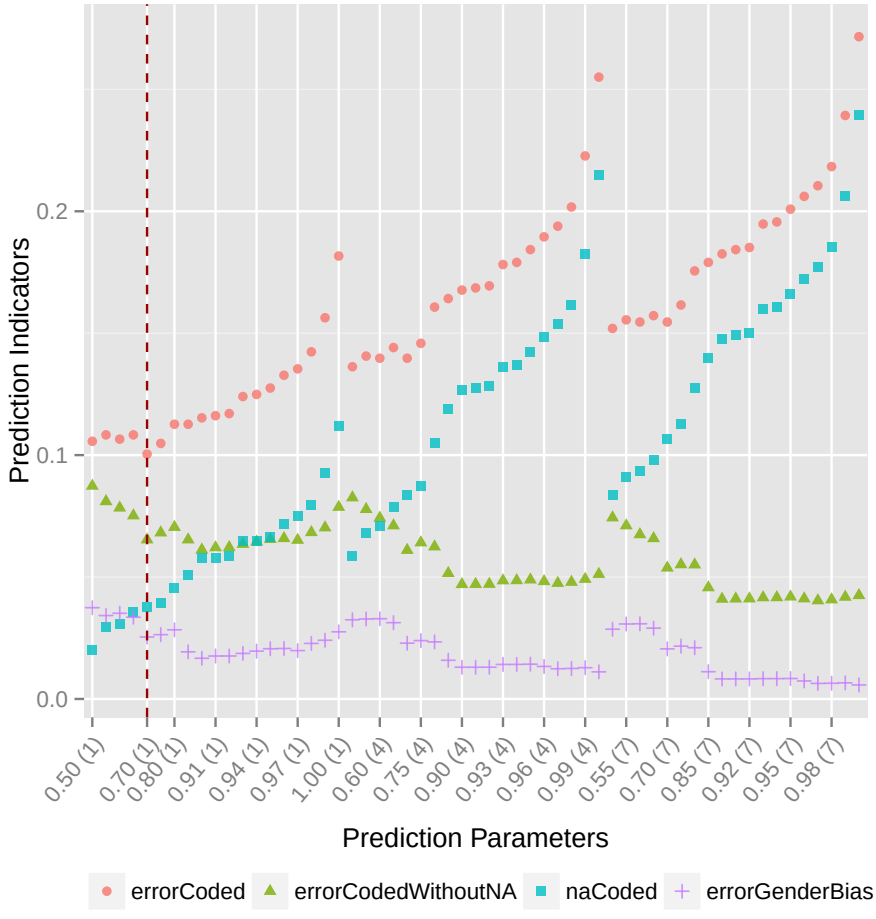


Figure 2: Effectiveness of gender prediction for given parameters from the title sample. The lowest `errorCoded` is achieved with probability = 0.70 and count = 1 (marked on the plot as a dashed red line), which also yields resonable low values of other error rates. The proportion of items with unpredicted gender (`naCoded`) increases periodically with the increase of the parameter values. The gender bias error rate (`errorGenderBias`) could be minimised at the cost of increasing the proportion of items with unpredicted gender. The final decision on which set of parameter values to use can be subjective and is dependent on the importance of each indicator in a given analysis.

As Jiang and Simon (2007) summarised, the LOO bootstrap is a smoothed version of the LOO cross-validation. Bootstrap samples differentiate among each other more than the original LOO sets. Moreover, for each item, the LOO bootstrap method averages the errors from the multiple predictions made on the bootstrap samples. As a result, the LOO bootstrap estimate has a much smaller variability than the LOO cross-validation estimate.

On the other hand, in contrast to low bias cross-validation, the LOO bootstrap in some cases presents noticeable bias and tends to overestimate the true prediction error. The *.632+* estimator was proposed as a remedy to deal with this problem (Efron and Tibshirani, 1997).

In gender prediction, we can use these bootstrap methods, as they are implemented in **genderizeR** package in the `genderizeBootstrapError` function.

We can still train our prediction algorithm on a large grid of parameters, or we can focus on combinations that we suspect could yield different outcomes on different bootstrap samples. In our `titles` dataset, we found that parameters `prob >= 0.70` and `count >= 1` give us the lowest apparent error rate on coded items, but it is reasonable to suspect that the combination of `prob >= 0.50` and `count >= 1` also could give us the lowest error rate on some random samples.

```
R> counts <- 1
R> probs <- c(0.5, 0.7)
R> set.seed(42)
R> bootstrapErrors <- genderizeBootstrapError(
+    # parallel = TRUE,
+    x = titles[titles$genderCoded %in% c('female', 'male')]$title,
+    y = titles[titles$genderCoded %in% c('female','male')]$genderCoded,
+    givenNamesDB = givenNamesDB_titles,
+    probs = probs,
+    counts = counts,
+    num_bootstraps = 50)
R> t(as.data.frame(bootstrapErrors))

                     [,1]
apparent         0.1004367
loo_boot         0.1037184
errorRate632plus 0.1025251
```

All bootstrap errors are calculated only on items with gender labels and predictions. The `apparent` value presents the underestimated *apparent error rate*; the `loo_boot` value provides the overestimated *LOO bootstrap error rate* and the `errorRate632plus` value gives *.632+* estimator that provides the best estimation of prediction error rate.

ROC and AUC

In order to plot the ROC curve or calculate AUC (Fawcett, 2003) for our predictions on the `title` sample, we can use the R package **ROCR** (Sing et al., 2005), but the datasets need to be prepared first. We need to have a data frame that shows which title has manually coded female (or male) gender types with a column of corresponding probabilities predicting these genders. To do that, we need to combine the original dataset with the prediction and corresponding gender data.

```
R> genderizedTitles_output <-  genderize(x = titles[['title']],
+                                        genderDB = givenNamesDB_titles,
+                                        progress = FALSE)
R> genderizedTitles <- cbind(as.data.frame(titles),
+                            as.data.frame(genderizedTitles_output))
R> genderizedTitles <-
+    left_join(x = genderizedTitles[, names(genderizedTitles) != 'gender'],
+              y = givenNamesDB_titles, by = c("givenName" = "name"))
R> example <- rownames(genderizedTitles[c(3, 9, 25, 27, 29, 37), ])
R> genderizedTitles %>%
+    dplyr::select(title, genderCoded, givenName,
+    probability, count, gender) %>%
+    filter(rownames(.) %in% example)

                                                                  title
1                      (Jacqueline) Nancy Mary Adams, CBE, QSO 1926-2007
2                                2005 R W P King Award - Robert J. Adams
```

```
3                                    A master potter (Josiah Wedgwood)
4 A musician without retirement - Claus Bantzer ceases and makes it again
5                   A pioneer in the world of advertising, Armando Testa
6                                    A tribute to Jean-Michel Quinodoz
  genderCoded givenName probability  count gender
1      female      mary        1.00  54051 female
2        male    robert        1.00 100216   male
3        male    josiah        0.98     43   male
4        male     claus        0.94     81   male
5        male   armando        0.99    329   male
6        male      jean        0.52  26799   male
```

In the next step, we can compute our vector of probabilities that we use to predict a given gender for each item.

```
R> d <- genderizedTitles
R> d[is.na(d[['gender']]), ][['gender']] <- 'unknown'
R> d <- d[d[['genderCoded']] %in% c('female', 'male', 'unknown' ), ]
R> d <- d[d[['gender']]  %in% c('female', 'male', 'unknown' ), ]
R> d$labels <- ifelse(d[['genderCoded']] == 'female', 1, 0)
R> d$pred <- ifelse(d[['gender']] == 'female',
+                   as.numeric(d[['probability']]),
+                   1-as.numeric(d[['probability']]))
R> d %>% dplyr::select(title, genderCoded, labels, pred) %>%
+    filter(rownames(.) %in% example)

                                                                    title
1                      (Jacqueline) Nancy Mary Adams, CBE, QSO 1926-2007
2                                2005 R W P King Award - Robert J. Adams
3                                    A master potter (Josiah Wedgwood)
4 A musician without retirement - Claus Bantzer ceases and makes it again
5                   A pioneer in the world of advertising, Armando Testa
6                                    A tribute to Jean-Michel Quinodoz
  genderCoded labels pred
1      female      1 1.00
2        male      0 0.00
3        male      0 0.02
4        male      0 0.06
5        male      0 0.01
6        male      0 0.48
```

Now we can plot the ROC curve and calculate the AUC.

```
R> library('ROCR')
R> pred <- prediction(labels = d[['labels']], predictions = d[['pred']])
R> perf <- performance(pred, measure = 'tpr', x.measure = 'fpr')
R> # area under the curve (AUC)
R> unlist(performance(pred, measure = 'auc')@y.values)

[1] 0.9186008
```

The AUC value is high (0.92), so we can conclude that the gender prediction algorithm performs well on this sample dataset. We can also treat the AUC measure as a general measure of predictiveness (Fawcett, 2003) and use it for comparison of different prediction methods.

Brier Score

Classification errors can be misleading metrics of gender prediction effectiveness, especially in the case when we have a great disproportion of female and male labels in the dataset. For example, in our `titles` dataset we have 87% male titles. If we predict male for each item in the dataset, our classification error will be only 13% due to small true proportion of female items.

```
R> titlesCoded <- titles[titles[['genderCoded']] %in% c('female', 'male'), ]
R> cbind('N' = table(titlesCoded[['genderCoded']]),
+        '%' = round(prop.table(table(titlesCoded[['genderCoded']])) * 100, 0))
```

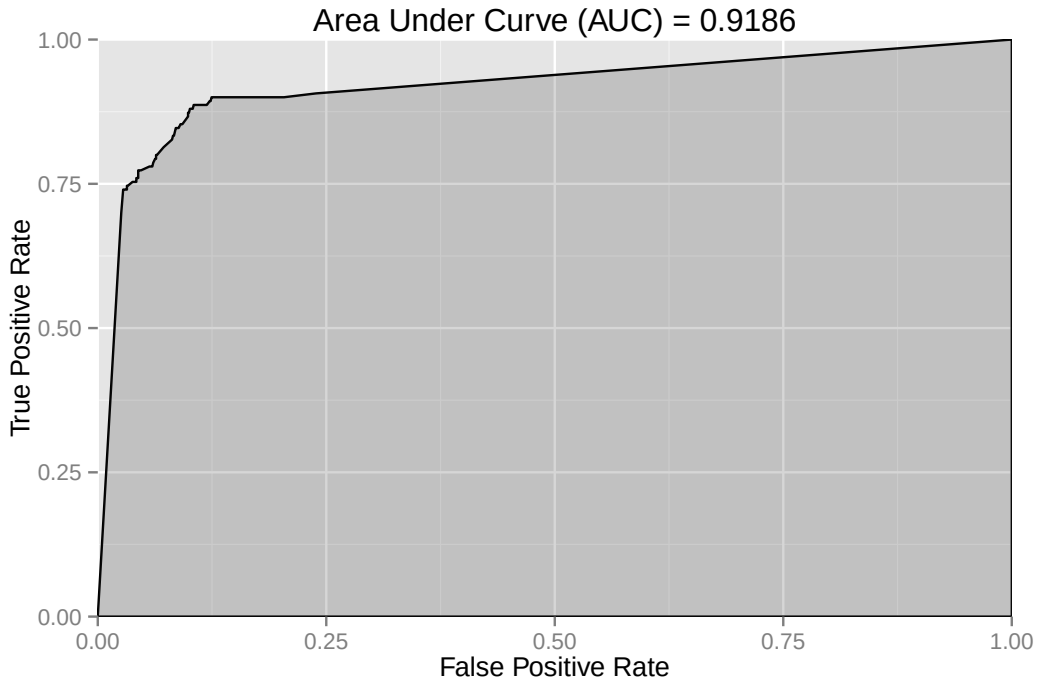


Figure 3: ROC curve for predicting female gender for the `titles` dataset. Large AUC suggests good prediction accuracy.

```
        N  %
female 151 13
male   994 87
R> indicators <-
+    classificationErrors(labels = titlesCoded[['genderCoded']],
+                         predictions = rep('male',
+                                           NROW(titlesCoded[['genderCoded']])))
R> unlist(indicators['errorCoded'])

errorCoded
 0.1318777
```

Predicting the most frequent class on every occasion in order to minimise the classification error rate is not an appropriate scoring rule and the prediction efficiency metrics should be penalised for such practices. That is why we use the *Brier Score*, which originated from a verification system of weather forecasts (Brier, 1950) and is known as a more proper classification accuracy score for predictions based on probabilities.

The *Brier Score* was defined as the sum of squared differences between `labels` values (ex. 1 if an item is a female and 0 if it is not) and `pred` values (probability if an item is a female) divided by the number of all items. The R package **verification** (NCAR – Research Applications Laboratory, 2014) offers an implementation of the *Brier Score* in the `brier` function.

```
R> library('verification')
R> brier(obs = d[['labels']], pred =  d[['pred']])[['bs']]

[1] 0.06577986
```

If an item's gender is correctly predicted with probability equal to 1, then the *Brier Score* is 0, and this is the best possible score. The worst score is 1.

We can also calculate presented and other metrics using a simulated bogus prediction algorithm. Such algorithm predicts the majority class (male labels) for all items. Both AUC and *Brier Score* are penalised since the bogus algorithm has just been guessing the gender using the most frequent class in this sample dataset.

Characteristics of prediction methods and prediction efficiency indicators	West et al. (2013)	Larivière et al. (2013)	**genderizeR** package
Source of first names data	US SSA	US Census	genderize.io
Extracted full first names	Yes	Yes	No
Gender probability threshold	0.95	0.91	0.50
Names count threshold	5	7200	1
Classification error rates (%):			
coded error rate	32.49	36.84	7.09
coded error rate without NA values	1.67	2.13	3.79
net gender bias error	1.67	2.13	1.42
Accuracy scores:			
AUC	0.926	0.920	0.927
Brier Score	0.099	0.109	0.097
Unpredicted gender (%):			
of all authorships	84.40	85.23	47.71
of authorships with full first names	31.56	35.22	4.65
of manually coded gender only	31.35	35.47	3.43

Table 3: Comparison of the effectiveness of gender prediction methods.

```
R> pred <- prediction(labels = d[['labels']],
+                     predictions = rep(0, NROW(d[['labels']])))
R> unlist(performance(pred, measure = 'auc')@y.values)

[1] 0.5
R> brier(obs = d[['labels']],
+        pred = rep(0, NROW(d[['labels']])))[['bs']]

[1] 0.12119
```

The comparison of methods

To compare different approaches of gender prediction, we need to reproduce those methods on the same sample that is drawn from the population of items with similar characteristics. We use an `authorships` dataset as such a sample, since predicting the gender of authors of the articles was the goal of two previously described studies.

Because the US SSA and the US Census data is available in the R packages, we can easily reproduce methods based on such data with parameters that were chosen by the authors of the studies. However, in the case of the method described by Larivière et al. (2013a), we can only reproduce gender prediction utilising the US Census data. Scraping and parsing other webpages or manually coding gender are highly resource intensive and our goal is to find a resource effective method. Nevertheless, we can later reproduce a confusion matrix from the study and try to estimate the real error for this mixed method.

The efficiency indicators of gender prediction for all three methods are presented in Table 3. For the third proposed method, we had not extracted first names from the authors' full names, and we let the `findGivenNames` function try to do this automatically.

As shown in Table 3, our proposed **method based on *genderize.io* API outperforms methods based on the US SSA and US Census data with the parameters set by the authors of these studies**. The percentage of items with unpredicted gender and *Brier Score* are clearly the lowest among all analysed methods.

It should be noted that using data sources other than the US Census with manual coding as done by Larivière et al. (2013a) would probably improve the proportion of items with predicted gender in our sample. However, it is difficult to assess the effectiveness of such a resource-consuming mixed method without comparable prediction metrics. The authors did not explicitly present the confusion matrix from their validation study, but we can recreate the matrix using materials from the analysed

	female	male	unisex	unknown	initials
female	146	4	2	24	48
male	22	267	9	51	156
unknown	35	178	10	9	39

Table 4: Recreated confusion matrix from Lariviere et al.'s (2013b) study with manually coded values (in rows) and automatically assigned values (in columns). The columns unknown, initials, and unisex from the recreated confusion matrix could be merged to one unknown column since the gender is unpredicted for these items.

studies, such as Table S3 *Number and percentage of distinct papers and of author-papers assigned a gender* and Table S6 *Percent male and female in each category* from the original paper (Larivière et al., 2013a, p. 6). From those original tables, we know the proportions of authorships classified as *female*, *male*, *unisex*, *unknown* and *initials*. The authors randomly sampled five samples of 1000 authorships from each category and manually coded whether those items were *female*, *male*, or *unknown* (Larivière et al., 2013a). From those known proportions, we can recreate the confusion matrix that sums to 1000 for a better perception (see Table 4).

Based on the recalculated data, the prediction indicators from the recreated confusion matrix are even worse than those in our comparative study where only the US Census data were used. The *coded error rate* is high and equals **43.3%**, since the proportion of manually coded items with unpredicted gender is also quite high **39.8%**. Moreover, the *gender bias error rate* in the original study is worse than in our comparative study based only on the US Census data, which suggests that **using additional data sources, manual coding, or unisex category will not necessarily increase the proportion of items with predicted gender and can also contribute to the bias of gender proportion estimates**.

Discussion

We cannot be sure how our method would perform on the same large datasets as in the described studies. Those studies were based on many non-English names and the *genderize.io* database has started gathering data mainly from public social profiles from the US and English-speaking countries, although it is still growing every minute and incorporating new data from other countries and languages.

The drawback of this data source is that people can put many different terms in their social network profiles that are not their first names. This could generate noise in the *genderize.io* database that could disturb gender prediction and introduce some bias if there are many instances of the same non-first-name terms used. Surprisingly, such crowd-sourced data work even better in our comparison study than very reliable and official data sources. In addition, we always can use a subset of the database with most reliable records.

We could also search for first names of papers' authors by taking into account the language in which they published their papers. Such an approach could be misleading, as many scientists publish papers in English, although it is not their native language. Another approach is to consider the proportion of males or females for a particular first name, while considering a year of birth of the person in question (Mullen, 2014). This historical method could improve gender predictions, but such birth data are often not available, especially in the discussed bibliometrics studies. However, if a year of birth is available, we could combine the two sources and use US Census data for years of birth from 1789 to 1930 and SSA data for newer ones as proposed by Mullen (2014). The particular problem with historic data is that the gender associated with a given name can change over time and the association can also differ geographically (Blevins and Mullen, 2015). The functions in the **genderizeR** package assume that we are looking for gender prediction based on global and contemporary gender-name associations.

The code in the **genderizeR** package works fast enough for many research purposes on datasets with more than 200 000 records of articles and their authors. Checking 108 023 unique terms through the *genderize.io* API took 45 minutes (2395.2 terms per minute). I have not tested and optimised the code for really large datasets yet, and by large datasets I mean millions of papers or authorships. However, some functions in the package are already implemented in parallel versions and can be used on larger datasets. I have already been experimenting with more efficient solutions from several other packages like **data.table** (Dowle et al., 2014), **dplyr** (Wickham and Francois, 2014), **stringr** (Wickham, 2012), **tm** (Feinerer and Hornik, 2014), and others. I am aware that many improvements could be made to the package and I am open for suggestions or contributions to the *GitHub* repository (`https://github.com/kalimu/genderizeR`).

The interesting prospect that can be explored in the future is the *genderizing* of some larger text corpuses. As we automatically try to ascribe gender to a title of a biographical article, we can additionally try to find first names in a larger text corpus and count how many references there are to females and males. It could be helpful, for example, in literature or media analysis.

Conclusion

Gender prediction is not as simple as it sometimes seems to be. In our data the gender category is not explicitly available for the researcher as it often is from, for example, a traditional survey question. When the name of a person in question is known, we can try to predict her or his gender based on the first name, even if we do not know which part of the record is a first name. This has many potential applications from big data analysis of authors of scientific papers to commercial applications where we can customise a commercial offer based on someone's gender without directly asking him or her about that.

The interest in gender identification seems to be increasing as we can see it in the studies done in recent years and in new IT tools that have emerged recently, such as *genderize.io*, R packages or code extensions for Ruby or Python that utilise gender data sources like *genderize.io* (Strømgren, 2015a).

New tools and a variety of open data sources (from the US Census, US SSA, and others) expand the possibilities of gender analysis, especially when they are in easy-readable machine formats. However, there is a shortage of proper validation studies that could show how effective the methods are that have been used for gender predictions. There is also a need to more explicitly present report prediction efficiency metrics that can be comparable with other methods across different studies.

From comparison of the three methods in this study, we can now assume that using *genderize.io* API as a gender data source is probably the best currently available approach. However, the outcomes could vary in different contexts, so it is always recommended to perform a validation study of different methods and compare a set of predictions metrics in order to select the most effective approach. Moreover, one should be cautious when using combined data sources or human coders as that can also have an adverse effect on some indicators of gender prediction efficiency.

Acknowledgements

This work is a result of many discussions with Marcin Kozak and Olesia Iefremova about different problems in bibliometrics studies, some of which the **genderizeR** package is intended to resolve. It would not exist without encouragement from Professor Kozak and his challenging questions (for which I am really very thankful). I would also like to thank Olesia Iefremova for manually checking and coding genders for over one thousand titles of biographical articles (which is a quite tedious and time-consuming activity). Last but not least, Casper Strømgren should be given special acknowledgements for practical implementation of the simple yet powerful idea of using crowd-sourced, publicly available data for gender prediction.

Bibliography

Web of Science, 2014. URL http://apps.webofknowledge.com. [p22]

C. Blevins and L. Mullen. Jane, John ... Leslie? A historical method for algorithmic gender prediction. *Digital Humanities Quarterly*, 9, 2015. URL http://www.digitalhumanities.org/dhq/vol/9/3/000223/000223.html. [p17, 35]

G. W. Brier. Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78(1): 1–3, 1950. doi: 10.1175/1520-0493(1950)078. [p30, 33]

M. Dowle, T. Short, S. Lianoglou, and A. Srinivasan. *data.table: Extension of data.frame*, 2014. URL http://CRAN.R-project.org/package=data.table. R package version 1.9.2, with contributions from R. Saporta and E. Antonyan. [p35]

B. Efron. Estimating the error rate of a prediction rule: Improvement on cross-validation. *Journal of the American Statistical Association*, 78(382):316–331, 1983. doi: 10.2307/2288636. [p19, 30]

B. Efron. How biased is the apparent error rate of a prediction rule? *Journal of the American Statistical Association*, 81(394):461–470, 1986. doi: 10.2307/2289236. [p30]

B. Efron and R. Tibshirani. Improvements on cross-validation: The .632 bootstrap method. *Journal of the American Statistical Association*, 92(438), June 1997. [p30, 31]

T. Fawcett. ROC graphs: Notes and practical considerations for data mining researchers. Technical report, Intelligent Enterprise Technologies Laboratory. Hawlett-Packard Laboratories, Palo Alto, CA, USA, 2003. [p30, 31, 32]

I. Feinerer and K. Hornik. *tm: Text Mining Package*, 2014. URL http://CRAN.R-project.org/package=tm. R package version 0.6. [p21, 35]

I. Feinerer, K. Hornik, and D. Meyer. Text mining infrastructure in R. *Journal of Statistical Software*, 25 (5):1–54, Mar. 2008. URL http://www.jstatsoft.org/v25/i05/. [p21]

W. Jiang and R. Simon. A comparison of bootstrap methods and an adjusted bootstrap approach for estimating the prediction error in microarray classification. *Statistics in Medicine*, 26(29):5320–5334, Dec. 2007. doi: 10.1002/sim.2968. [p31]

V. Larivière, C. Ni, Y. Gingras, B. Cronin, and C. R. Sugimoto. Supplementary information to "Bibliometrics: Global gender disparities in science" (Comment in Nature 504, 211–213; 2013). *Nature*, 504, 12 2013a. [p17, 18, 20, 34, 35]

V. Larivière, C. Ni, Y. Gingras, B. Cronin, and C. R. Sugimoto. Bibliometrics: Global gender disparities in science. *Nature*, 504:211–213, 12 2013b. doi: 10.1038/504211a. [p17]

L. Mullen. *gender: Predict Gender From Names Using Historical Data*, 2014. URL https://github.com/ropensci/gender. [p19, 35]

NCAR – Research Applications Laboratory. *verification: Weather Forecast Verification Utilities.*, 2014. URL http://CRAN.R-project.org/package=verification. R package version 1.41. [p33]

J. A. Ramey. *sortinghat*, 2013. URL http://CRAN.R-project.org/package=sortinghat. R package version 0.1. [p21]

T. W. Rinker. *qdap: Quantitative Discourse Analysis Package*. University at Buffalo/SUNY, Buffalo, New York, 2013. URL http://github.com/trinker/qdap. Version 2.2.1. [p18]

T. Sing, O. Sander, N. Beerenwinkel, and T. Lengauer. ROCR: Visualizing classifier performance in R. *Bioinformatics*, 21(20):3940–3941, 2005. URL http://rocr.bioinf.mpi-sb.mpg.de. [p31]

Social Security Administration. Official social security website, 2015. URL http://www.ssa.gov/oact/babynames/limits.html. [p19]

C. Strømgren. genderize.io, 2015a. URL http://genderize.io. [p19, 20, 36]

C. Strømgren. store.genderize.io, 2015b. URL https://store.genderize.io. [p19]

United States Census Bureau. Frequently occurring surnames from Census 1990 – Names files, 2015a. URL http://www.census.gov/topics/population/genealogy/data/1990_census/1990_census_namefiles.html. [p18]

United States Census Bureau. Frequently occurring surnames from the Census 2000, 2015b. URL http://www.census.gov/topics/population/genealogy/data/2000_surnames.html. [p18]

K. Wais. *genderizeR: Gender Prediction Based on First Names*, 2016. URL http://CRAN.R-project.org/package=genderizeR. R package version 2.0.0. [p20, 21]

J. D. West, J. Jacquet, M. M. King, S. J. Correll, and C. T. Bergstrom. The role of gender in scholarly authorship. *PLOS ONE*, 8, 7 2013. [p17, 19, 20]

H. Wickham. *stringr: Make It Easier To Work With Strings*, 2012. URL http://CRAN.R-project.org/package=stringr. R package version 0.6.2. [p21, 35]

H. Wickham. *babynames: US Baby Names 1880–2013*, 2014. URL http://github.com/hadley/babynames. R package version 0.1. [p19]

H. Wickham and R. Francois. *dplyr: A Grammar of Data Manipulation*, 2014. URL http://CRAN.R-project.org/package=dplyr. R package version 0.3.0.2. [p35]

Kamil Wais
University of Information Technology and Management in Rzeszow
Sucharskiego 2, 35-225 Rzeszow
Poland
kamil.wais@gmail.com

Conditional Fractional Gaussian Fields with the Package FieldSim

by Alexandre Brouste, Jacques Istas and Sophie Lambert-Lacroix

Abstract We propose an effective and fast method to simulate multidimensional conditional fractional Gaussian fields with the package **FieldSim**. Our method is valid not only for conditional simulations associated to fractional Brownian fields, but to any Gaussian field and on any (non regular) grid of points.

Introduction

Rough phenomena arise in texture simulations for image processing or medical imaging, natural scenes simulations (clouds, mountains) and geophysical morphology modeling, financial mathematics, ethernet traffic, *etc.* Some are time-indexed, some others, like texture or natural scene simulations, should be indexed by subsets of the Euclidean spaces $\mathbb{R}^2$ or $\mathbb{R}^3$. Recent data (as the Cosmic Microwave Background or solar data) are even indexed by a manifold.

The fractional Brownian motion (fBm), introduced by Kolmogorov (1940) (and developed by Mandelbrot and Van Ness 1968) is nowadays widely used to model this roughness. Fractional Brownian motions have been extended in many directions: higher dimensions with fields, anisotropy, multifractionality, *etc.* This paper is devoted to a simulation method for conditional Gaussian fields. This could improve, in the future, natural scene simulations by fixing for instance the valleys.

The simulation of fractional Gaussian processes is not difficult in dimension one (see a review of Coeurjolly 2000). Let us recall the numerical complexity of some classical methods: the Cholesky method has a complexity of $O(N^3)$ where N is the size of the simulated sample path. For specific stationary processes (on a regular grid) the Levinson's algorithm has a complexity of $O(N^2 \log N)$ and the Wood and Chan algorithm (see Wood and Chan 1994) a complexity of $O(N \log N)$.

In higher dimensions, the Wood and Chan method has been extended to stationary increments fields with the Stein's method (Stein, 2002) ; the fractional Brownian field can therefore be simulated on a regular grid of the plane. For general Gaussian fields on a general discrete grid, the Cholesky method is costly and exact simulations are no longer tractable. Approximate methods have been intensively developed (midpoint, Peitgen and Saupe 1988; turning bands, Yin 1996; truncated wavelet decomposition) but for specific fields. On manifolds, simulation procedures based on truncated series of eigenfunctions of the Laplace-Beltrami operator are discussed in Gelbaum and Titus (2014).

Our approach, presented in Brouste et al. (2007, 2010), is based on a 2-steps method with an exact simulation step plus a refined fast step, that is an improvement of the midpoint method. It has been implemented in the **FieldSim** package (Brouste and Lambert-Lacroix., 2015). The fieldsim simulation method can be applied to general Gaussian processes on general simulation grids (regular and non regular) on Euclidean spaces and even on some manifolds (see Figure 1). It is worth mentioning that another package, **RandomFields** (Schlather et al., 2016), allows the simulation of a large class of random fields such as Gaussian random fields, Poisson fields, binary fields, chi-square fields, t fields and max-stable fields (see Schlather et al. 2015). In **RandomFields**, conditional random fields (which are the purpose of the present paper) are given for a wide range of spatial and spatio-temporal Gaussian random fields. Some of the default models of the **FieldSim** package cannot be simulated with the help of default models of the **RandomFields** package. Nevertheless, it is still possible to simulate them with the `RMuser()` and `RFsimulate()` commands of the **RandomFields** package. It may be noted that the **FieldSim** package does not allow for the simulation of more than the **RandomFields** package. **FieldSim** package is an alternative in which the underlying methods of simulation are generic.

We propose here to adapt the **FieldSim** package to conditional simulations. Definitions and notation will be introduced in the following section with the "process" class, the `setProcess` procedure and the `fieldsim` procedure. The `fieldsim` procedure adapted to conditional Gaussian fields is described in the next section. Simulations with the package **FieldSim** are presented in the last section.

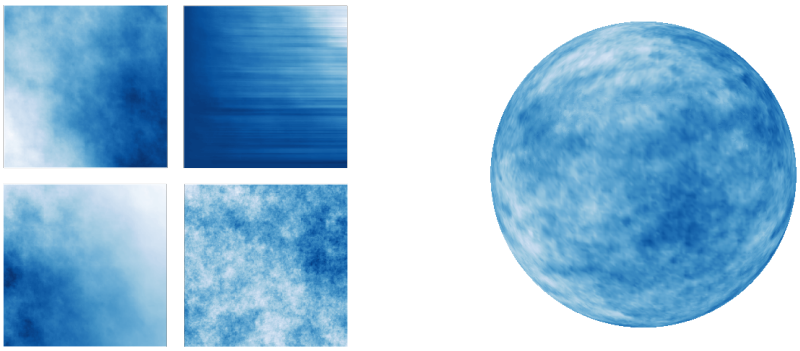


Figure 1: On the left: fractional Brownian field (top-left), multifractional Brownian field (bottom-left), fractional Brownian sheet (top-right) and hyperbolic fractional Brownian field (bottom-right); on the right: fractional Brownian field on the sphere.

Notation and preliminaries

Fractional Gaussian fields

Let d be a positive integer and $X(\cdot) = \left\{X(M), M \in \mathbb{R}^d\right\}$ be a real valued non stationary field with zero mean and second order moments. It is worth emphasizing that we consider in this paper the metric space $\mathbb{R}^d$ with the Euclidean norm but the method can be generalized to a smooth and complete Riemannian manifold equipped with its geodesic distance (Brouste et al., 2010).

The covariance function $R(\cdot,\cdot)$ is defined by:

$$R(M_1, M_2) = \operatorname{cov}\left(X(M_1), X(M_2)\right), \quad M_1,\ M_2 \in \mathbb{R}^d.$$

This function is nonnegative definite (n.n.d.). Conversely, for any n.n.d. function $R(\cdot,\cdot)$, there exists an unique centered Gaussian field of second order structure given by $R(\cdot,\cdot)$.

Different classical fractional Gaussian fields have been simulated to illustrate the **FieldSim** package in Brouste et al. (2007, 2010). In the sequel, M and M' are two points of $\mathbb{R}^d$ and $\|\cdot\|$ is the usual norm on $\mathbb{R}^d$, $d = 1, 2$. We can cite:

1. The standard fractional Brownian fields are defined through their covariance function (e.g., Samorodnitsky and Taqqu 1994):

$$R(M, M') = \frac{1}{2}\left(\|M\|^{2H} + \|M'\|^{2H} - \|M - M'\|^{2H}\right),$$

   where the Hurst parameter H is real in $(0, 1)$.

2. The standard multifractional Brownian fields are defined through their covariance function (see Peltier and Levy-Véhel 1996; Benassi et al. 1997):

$$R(M, M') = \alpha(M, M')\left(\|M\|^{\tilde{H}(M,M')} + \|M'\|^{\tilde{H}(M,M')} - \|M - M'\|^{\tilde{H}(M,M')}\right),$$

   where

$$\begin{aligned}
\tilde{H}(M, M') &= H(M) + H(M'),\\
\alpha(M, M') &= \frac{C\left(\frac{H(M)+H(M')}{2}\right)^2}{2C\left(H(M)\right)C\left(H(M')\right)},\\
C(h) &= \left(\frac{\pi^{\frac{d+1}{2}}\Gamma\left(h+\frac{1}{2}\right)}{h\sin\left(\pi h\right)\Gamma\left(2h\right)\Gamma\left(h+\frac{d}{2}\right)}\right)^{\frac{1}{2}},
\end{aligned}$$

   and the Hurst parameter is a continuous function $H : \mathbb{R}^d \longrightarrow (0, 1)$, where Γ is the usual Gamma function.

3. The standard fractional Brownian sheets are defined through their covariance function (see

Kamont 1996):

$$R(M,M') = \frac{1}{2^d}\prod_{i=1}^{d}\left\{|M_i|^{2H_i} + |M_i'|^{2H_i} - |M_i - M_i'|^{2H_i}\right\},$$

where $(H_1,\ldots,H_d)$ stands for the multivariate Hurst index in $\mathbb{R}^d$, $0 < H_i < 1$.

4. The anisotropic fractional Brownian fields are defined through their covariance function (see Bonami and Estrade 2003):

$$R(M,M') = v_H(M) + v_H(M') - v_H(M - M'),$$

where the variogram

$$v_H(x) = 2^{2H-1}\gamma(H)C_{H,\vartheta_1,\vartheta_2}(x)\|x\|^{2H},$$

with $H \in (0,1)$, $\gamma(H)$ depends explicitly on H and $C_{H,\vartheta_1,\vartheta_2}(.)$ implies incomplete Beta functions and two constants $-\frac{\pi}{2} \leq \vartheta_1 < \vartheta_2 \leq \frac{\pi}{2}$.

The FieldSim package

In the new version 3.2 of the package **FieldSim**, new features have been added. The most important add is the "process" class and the `setProcess` function.

An object of class "process" has different slots:

- The `name` of the process. Several names are reserved for classical fractional Gaussian processes: see Table 1 for details. `"cond"` is used for all kind of conditional simulations (see further).
- The slot `values` stores the values of the process on the simulation (and visualization) grid.
- An object of class "manifold" which is the Riemannian manifold on which the process is lying; an object of the class "manifold" has four slots:
  - `name` which is the name of the manifold we consider. The name `"line"`, `"plane"`, `"sphere"` and `"hyperboloid"` are taken for the eponymous manifolds.
  - `atlas` which is the union of discretized domains that cover the manifold (must be a matrix where the number of rows is the dimension of the space where the manifold lives).
  - `distance` which is the distance considered on the manifold.
  - `origin` which is the origin considered on the manifold (must be a point on the manifold).

  The setter `setManifold` permits the user to create an object of class "manifold" with all its slots. This class is already described in Brouste et al. (2010).
- The slot `covf` which contains the covariance function of the Gaussian process.
- The slot `parameter` which contains all the parameters associated to the covariance function of the process. Here are the classical parameters associated to the classical process.

All the examples presented can be defined with the `setProcess` command (see Table 1). With the following command, the user can set a fBm with Hurst parameter 0.7 on a regular grid of the interval $[0,1]$ (of size 256).

```
R> linefBm <- setProcess("fBm-line", 0.7)
R> str(linefBm)

Formal class 'process' [package "FieldSim"] with 7 slots
  ..@ name     : chr "fBm"
  ..@ values   : num 0
  ..@ manifold :Formal class 'manifold' [package "FieldSim"] with 4 slots
  .. .. ..@ name    : chr "line"
  .. .. ..@ atlas   : num [1, 1:256] 0 0.00392 0.00784 0.01176 0.01569 ...
  .. .. ..@ distance:function (xi, xj)
  .. .. ..@ origin  : num [1, 1] 0
  ..@ covf     :function (xi, xj)
  ..@ parameter: num 0.7
  ..@ values2  : num 0
  ..@ manifold2:Formal class 'manifold' [package "FieldSim"] with 4 slots
  .. .. ..@ name    : chr "line"
  .. .. ..@ atlas   : num [1, 1:256] 0 0.00392 0.00784 0.01176 0.01569 ...
  .. .. ..@ distance:function (xi, xj)
  .. .. ..@ origin  : num [1, 1] 0
```

It is worth mentioning that the slot `values` is empty since there is no simulation done. Then as usual, the user can use the `fieldsim` function in order to simulate the Gaussian process associated to `covf` on the manifold grid defined in `manifold`.

```
R> fieldsim(linefBm)
```

In the `fieldsim` function, we can add the quantity `Ne`, the number of points of the grid to be simulated in the exact step, and `nbNeighbor`, the number of neighbors used in the refined step. By default, `Ne` is equal to the size of the grid given in `atlas`. The slot `values` are now set with the simulated values. There exist different visualization procedures to draw the results, for instance:

```
R> plot(linefbm, "default")
```

We recall that the discretization grids can be modified with the `setAtlas` command. Depending on the manifold, there are several types of grids: `"regular"`, `"random"` and `"visualization"`. For instance,

```
R> setAtlas(linefBm, "regular", 1000)
R> fieldsim(linefBm)
R> plot(linefBm, "default")
```

The `fieldsim` procedure for conditional Gaussian fields

In order to build conditional fractional Gaussian fields, we consider a conditioning set $\mathcal{N} = \{N_1, \ldots, N_k\}$, $N_i \in \mathbb{R}^d$, $i = 1, \ldots, k$, and the conditioning values $\mathbf{x} = (x_1, \ldots, x_k)^T \in \mathbb{R}^k$. Then we will say that $\widetilde{X}(\cdot) = \{\widetilde{X}(M), M \in \mathbb{R}^d\}$ is the conditional Gaussian field associated to the field $X(\cdot)$ (of covariance function R) and to the conditioning pair $(\mathcal{N}, \mathbf{x})$ if the finite dimensional laws of $\widetilde{X}(\cdot)$ is the same as the finite dimensional laws of $X(\cdot)$ given the event $\{(X(N_1), \ldots, X(N_k))^T =: \mathbf{X}_{\mathcal{N}} = \mathbf{x}\}$. We denote by $\widetilde{m}(\cdot)$ (resp. $\widetilde{R}(\cdot, \cdot)$) the mean (resp. covariance) function of the process $\widetilde{X}(\cdot)$. The following lemma allows us to determine $\widetilde{m}(\cdot)$ and $\widetilde{R}(\cdot, \cdot)$ according to $R(\cdot, \cdot)$ (sketch of proof is given in Piterbag 1996, Section A.1).

Lemma 1. *Let us consider the centered Gaussian vector* $\left(Y_1, Y_2, \mathbf{Z}^T\right)^T \in \mathbb{R} \times \mathbb{R} \times \mathbb{R}^k$ *with the covariance matrix*

$$\Sigma^2 = \begin{pmatrix} \mathbb{E}(Y_1^2) & \mathbb{E}(Y_1 Y_2) & \mathbb{E}(Y_1 \mathbf{Z}^T) \\ \mathbb{E}(Y_1 Y_2) & \mathbb{E}(Y_2^2) & \mathbb{E}(Y_2 \mathbf{Z}^T) \\ \mathbb{E}(\mathbf{Z} Y_1) & \mathbb{E}(\mathbf{Z} Y_2) & \mathbb{E}(\mathbf{Z}\mathbf{Z}^T) \end{pmatrix}.$$

Suppose that $\mathbb{E}(\mathbf{Z}\mathbf{Z}^T)$ *is invertible. Then the conditional law of* $(Y_1, Y_2)^T$ *given the event* $\{\mathbf{Z} = \mathbf{z} \in \mathbb{R}^k\}$ *is Gaussian with mean*

$$\widetilde{m} = \begin{pmatrix} \mathbb{E}(Y_1 \mathbf{Z}^T) \\ \mathbb{E}(Y_2 \mathbf{Z}^T) \end{pmatrix} \{\mathbb{E}(\mathbf{Z}\mathbf{Z}^T)\}^{-1} \mathbf{z}, \tag{1}$$

and covariance matrix

$$\widetilde{\Sigma}^2 = \begin{pmatrix} \mathbb{E}(Y_1^2) & \mathbb{E}(Y_1 Y_2) \\ \mathbb{E}(Y_1 Y_2) & \mathbb{E}(Y_2^2) \end{pmatrix} - \begin{pmatrix} \mathbb{E}(Y_1 \mathbf{Z}^T) \\ \mathbb{E}(Y_2 \mathbf{Z}^T) \end{pmatrix} \{\mathbb{E}(\mathbf{Z}\mathbf{Z}^T)\}^{-1} \begin{pmatrix} \mathbb{E}(\mathbf{Z} Y_1) & \mathbb{E}(\mathbf{Z} Y_2) \end{pmatrix}. \tag{2}$$

In the Gaussian field context, Lemma 1 allows us to write down an explicit expression of the mean function and the autocovariance function of the conditional Gaussian field associated to $R(\cdot, \cdot)$ and to $(\mathcal{N}, \mathbf{x})$. Let us put $Y_1 = X(M_1)$ and $Y_2 = X(M_2)$ the values of the field $X(\cdot)$ at points $M_1 \in \mathbb{R}^d$ and $M_2 \in \mathbb{R}^d$ respectively, and $\mathbf{Z} = \mathbf{X}_{\mathcal{N}} \in \mathbb{R}^k$. Therefore, all quantities in (1) and (2) can be expressed in terms of the autocovariance function R. Precisely,

$$\mathbb{E}(Y_i Y_j) = R(M_i, M_j), \quad (i, j) \in \{1, 2\}^2,$$

and

$$\mathbb{E}(Y_i \mathbf{Z}_\ell) = R(M_i, N_\ell), \quad i \in \{1, 2\}, \quad \ell = 1, \ldots, k.$$

Consequently, the mean function of the conditional Gaussian field is given by

$$\widetilde{m}(M) = \mathbb{E}(X(M)\mathbf{X}_{\mathcal{N}}^T)\{\mathbb{E}(\mathbf{X}_{\mathcal{N}}\mathbf{X}_{\mathcal{N}}^T)\}^{-1}\mathbf{x}, \quad M \in \mathbb{R}^d. \tag{3}$$

Then the autocovariance function of a conditional Gaussian field (using the $(1, 2)$-coordinate of Equation (2)) is given by

$$\widetilde{R}(M_1, M_2) = R(M_1, M_2) - \mathbb{E}(X(M_1)\mathbf{X}_{\mathcal{N}}^T)\{\mathbb{E}(\mathbf{X}_{\mathcal{N}}\mathbf{X}_{\mathcal{N}}^T)\}^{-1}\mathbb{E}(\mathbf{X}_{\mathcal{N}} X(M_2)). \tag{4}$$

For instance, for $k = 1$, we get

$$\widetilde{m}(M) = \frac{R(M, N_1)}{R(N_1, N_1)} x_1,$$

and

$$\widetilde{R}(M_1, M_2) = R(M_1, M_2) - \frac{R(M_1, N_1)R(M_2, N_1)}{R(N_1, N_1)}.$$

Let us recall that the goal of this paper is to give a procedure that yields discretization of the sample path of the conditional Gaussian field over a space discretization $\{\mathcal{S}_e, \mathcal{S}_r\}$ of $\mathbb{R}^d$ associated to the n.n.d. autocovariance function R and the conditioning set and values $(\mathcal{N}, \mathbf{x})$. In the sequel, we denote by $\widetilde{X}(\cdot)$ this sample path. Since the mean function (3) is known, we can consider the centered field $\overline{X}(\cdot) = \widetilde{X}(\cdot) - \widetilde{m}(\cdot)$. The fieldsim procedure for conditional Gaussian fields proceeds as follows.

Exact simulation step. Given a space discretization $\mathcal{S}_e$, a sample of a centered Gaussian vector $(\overline{X}(M))_{M \in \mathcal{S}_e}$ with covariance matrix $\widetilde{\mathbf{R}}$ given by $\{\widetilde{\mathbf{R}}\}_{i,j} = \widetilde{R}(M_i, M_j)$, $M_i, M_j \in \mathcal{S}_e$, is simulated. Here $\widetilde{R}$ is defined by (4). This simulation is obtained by an algorithm based on Cholesky decomposition of the matrix $\widetilde{\mathbf{R}}$.

Refined simulation step. Let $\mathcal{S}_r$ be the remaining space discretization. For each new point $M \in \mathcal{S}_r$ at which we want to simulate the field, $\overline{X}(M)$ is generated by using only a set of neighbors instead of all the simulated components (as in the accurate simulation step). Precisely, let $\mathcal{O}_M$ be a neighbors set of M (for the Euclidean distance) and $\mathcal{X}_{\mathcal{O}_M}$ be the space generated by the variables $X(M')$, $M' \in \mathcal{O}_M$. Let us remark that the neighbors set is defined with all the already simulated variables (in the accurate and refined simulation step). Let $X_{\mathcal{X}_{\mathcal{O}_M}}(M)$ be the best linear combination of variables of $\mathcal{X}_{\mathcal{O}_M}$ approximating $\overline{X}(M)$ in the sense that the variance of the innovation

$$\varepsilon_{\mathcal{X}_{N_M}}(M) = \overline{X}(M) - X_{\mathcal{X}_{\mathcal{O}_M}}(M),$$

is minimal. The new variable $\overline{X}(M)$ is obtained by

$$X_{\mathcal{X}_{\mathcal{O}_M}}(M) + \sqrt{Var(\varepsilon_{\mathcal{X}_{\mathcal{O}_M}}(M))}U,$$

where U is a centered and reduced Gaussian variable independent of the already simulated components. Note that the variable $X_{\mathcal{X}_{\mathcal{O}_M}}(M)$ and the variance $Var(\varepsilon_{\mathcal{X}_{\mathcal{O}_M}}(M))$ are completely determined by the covariance structure of the sequence $\overline{X}(M')$, $M' \in \mathcal{O}_M \cup \{M\}$.

Adding the mean. Finally, we compute $\widetilde{X}(M) = \overline{X}(M) + \widetilde{m}(M)$ for all $M \in \{\mathcal{S}_e, \mathcal{S}_r\}$.

For storage and computing time, the accurate simulation step must concern only a small number of variables whereas the second step can relate to a larger number of variables. That leads to an effective and fast method to simulate any Gaussian field.

It is worth mentioning that the `setProcess` command will check if $\{\mathbb{E}(\mathbf{X}_{\mathcal{N}}\mathbf{X}_{\mathcal{N}}^T)\}^{-1}$ exists for common conditional simulations.

Some examples of conditional fractional Gaussian fields

We focus, in this paper, on the conditional Gaussian fields associated to the previously mentioned fields but every other classical Gaussian field can be also simulated: standard bifractional Brownian motion, space-time deformed fractional Brownian motion, *etc.* (see Brouste et al. 2007). We also consider conditional simulations associated to fractional Gaussian fields on manifolds (hyperboloid and sphere) (see Brouste et al. 2010 for the covariance function definition).

The procedure `fieldsim` is extended to the conditional Gaussian fields. We can find the `setProcess` reference short-card in Table 1.

On the line

The fractional Gaussian processes on the line are fast to simulate.

Conditional simulations associated to fractional Brownian motion (fBm) and multifractional Brownian motion (mBm) and to the conditioning set $\mathcal{N} = \{1/2, 3/4, 1\}$ and conditioning values

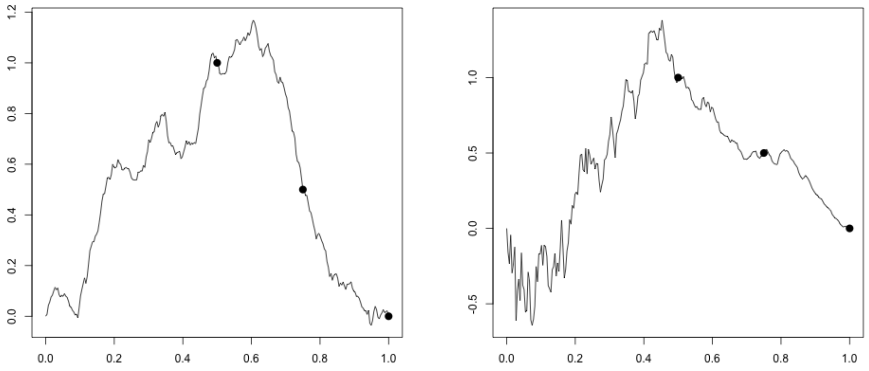


Figure 2: Conditional simulations associated to fractional Brownian motion and multifractional Brownian motion. The real time (resp. CPU time) in seconds is equal to 8.430 (resp. 0.043) for the fractional Brownian motion and 14.609 (resp. 0.111) for the multifractional Brownian motion.

$\mathbf{x} = \{1, 1/2, 0\}$ are illustrated on Figure 2. Here the Hurst exponent is $H = 0.7$ for the fBm and $H(t) = 0.3 + 0.6t$, $t \in [0, 1]$ for the mBm. The processes are simulated on a regular grid of 256 points of $[0, 1]$ with only an exact simulation step ($\mathcal{S}_r = \emptyset$).

They can be obtained with the `fieldsim` procedure. For instance, the mBm in Figure 2 is obtained with:

```
R> funcH <- function(x) 0.3 + x * 0.6
R> cond.mBm <- setProcess("cond-mBm-line",
+    list(Gamma = matrix(c(1/2, 1, 3/4, 0.5, 1, 0), 2, 3), par = funcH))
R> fieldsim(cond.mBm)
R> plot(cond.mBm)
```

In the simulation below, the points of the set $\mathcal{N}$ belong to the visualization grid. When this is not the case, the plot could show a failure for the conditioning in the region of high variability. To avoid this, it is possible to add the points of the set $\mathcal{N}$ to the visualization grid. For instance, in the previous example, to add the point $1/6$ to the visualization grid, we can use the following lines of code:

```
R> atlas.cond.mBm <- sort(c(cond.mBm@manifold@atlas[1, ], 1/6))
R> cond.mBm@manifold@atlas <- matrix(atlas.mBm, nrow = 1)
```

Another solution is to use finer grids which contain the points of the set $\mathcal{N}$.

On the plane

Conditional simulations associated to a fractional Brownian field (for $H = 0.9$) and multifractional Brownian field (for $H(\mathbf{t}) = 0.3 + 0.6t_1$) are illustrated in Figure 3. Conditional simulations associated to anisotropic fields (fractional Brownian sheet with $H_1 = 0.9$, $H_2 = 0.3$, anisotropic fractional Brownian field with $H = 0.7$, $\vartheta_1 = \frac{\pi}{6}$ and $\vartheta_2 = \frac{\pi}{3}$) are presented in Figure 4. For all the fields, we consider the following conditioning set

$$\mathcal{N} = \left\{ \left(1, \frac{k}{2^6+1}\right), \left(\frac{k}{2^6+1}, 1\right), k = 0, \ldots, 2^6+1 \right\},$$

and conditioning values $\mathbf{x} = \mathbf{0}$.

All the processes are simulated on a regular grid of 4096 points of $[0, 1]^2$ with 100 points for the exact simulation step and 3996 for the refined step (with 4 neighbors). For instance, the conditional Gaussian field associated to anistropic fractional Brownian field on $[0, 1]^2$ (see Figure 4) is given by

```
R> Ng <- 2^6 + 1
R> x <- seq(from = 0, to = 1, length = Ng)
R> G <- cbind(rbind(rep(1, Ng - 1), x[2:Ng], rep(0, Ng - 1)),
+    rbind(x[2:(Ng - 1)], rep(1, Ng - 2), rep(0, Ng - 2)))
R> condfBm2d <- setProcess("cond-afBf-plane",
+    list(Gamma = G, par = list(H = 0.7, theta1 = pi/6, theta2 = pi/3)))
R> setAtlas(condfBm2d, "visualization", 6)
R> fieldsim(condfBm2d, Ne = 100, nbNeighbor = 4)
R> plot(condfBm2d, theta = 120, phi = 30, expand = 0.5)
```

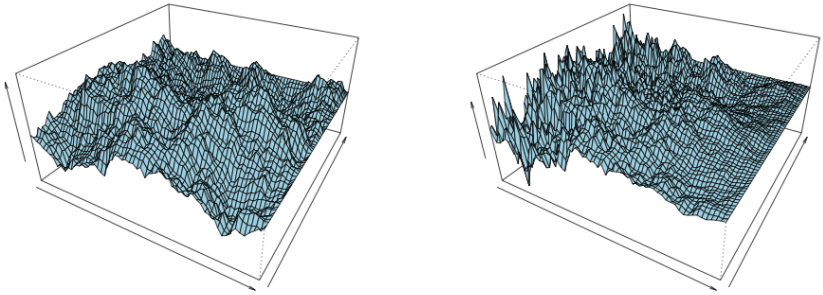


Figure 3: Conditional simulations associated to a fractional Brownian field (on the left) and a multifractional Brownian field (on the right). The real time (resp. CPU time) in seconds is equal to 1270.911 (resp. 6.769) for the fractional Brownian field and 1782.533 (resp. 9.953) for the multifractional Brownian field.

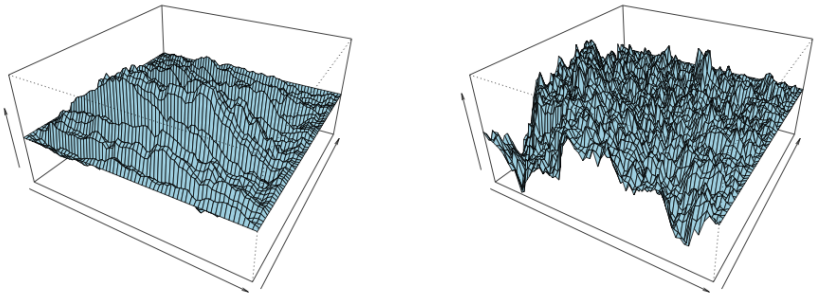


Figure 4: Conditional simulations associated to a fractional Brownian sheet (on the left) and an anisotropic fractional Brownian field. The real time (resp. CPU time) in seconds is equal to 728.605 (resp. 4.223) for the fractional Brownian sheet and 2995.644 (resp. 14.782) for the anisotropic fractional Brownian field.

It is worth emphasizing that, for a fixed size of the simulation grid, the simulation time of the `fieldsim` procedure depends on the number of conditioning points (see for instance the fractional Brownian field on Figures 3 and 6 for 129 and 39 conditioning points respectively). But this variation is small compared to the variation due to the size of the simulation grid.

On the hyperboloid and on the sphere

Conditional simulations can be extended to fractional Gaussian fields on manifolds associated to the fractional Brownian field on the hyperboloid with $H = 0.7$,

$$\mathcal{N} = \left\{(0,1,\sqrt{2}),\ (0,2,\sqrt{5})\right\}, \quad x = (5,-5),$$

and a conditional fractional Brownian field on the sphere with $H = 0.4$,

$$\mathcal{N} = \left\{(0,0,1),\ \left(\frac{1}{2},0,\frac{\sqrt{3}}{2}\right)\right\}, \quad x = (5,-5).$$

The two processes are simulated on a regular grid of 5400 points of $\mathbb{R}^3$ with 100 points for the exact simulation step and 5300 for the refined step (with 4 neighbors).

The conditional simulations associated to the fractional Brownian field on the sphere (see Figure 5) are obtained with

```
R> Gamma <- matrix(c(0, 0, 1, 5, 0.5, 0, sqrt(3)/2, -5), 4, 2)
R> sphere.cond.fBm <- setProcess("cond-fBm-sphere", list(Gamma = Gamma, par = 0.4))
R> setAtlas(sphere.cond.fBm, "visualization", 30)
R> fieldsim(sphere.cond.fBm, Ne = 100, nbNeighbor = 4)
R> plot(sphere.cond.fBm)
```

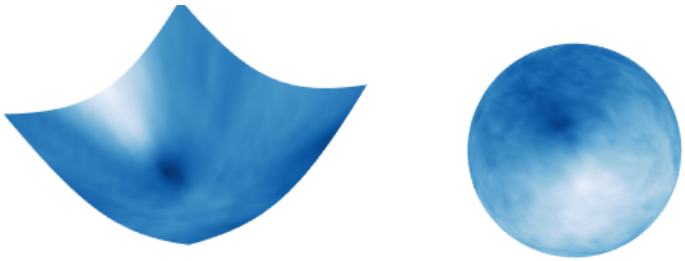


Figure 5: Conditional simulations associated to the fractional Brownian field on the hyperboloid and on the sphere. The real time (resp. CPU time) in seconds is equal to 54.567 (resp. 0.293) for the hyperboloid and 188.807 (resp. 14.216) for the sphere.

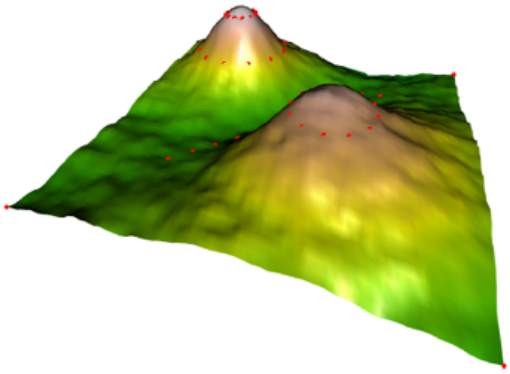


Figure 6: Natural scene simulation. Here a 65×65 regular grid fractional Brownian field of Hurst parameter $H = 0.8$ with 39 conditioning points (in red) is simulated. The real time (resp. CPU time) in seconds is equal to 563.754 (resp. 3.550).

Conclusion and perspectives

We propose a generic method to simulate multidimensional conditional fractional Gaussian fields.

Our method is valid for any Gaussian field and on any (non regular) grid of points as soon as the covariance function is available. This method is constructed to be universal (conditional simulation, simulation on a manifold) and is consecutively not as fast as other methods defined for specific fields. In the near future, the **FieldSim** package should also possess such specific methods.

Our method is adapted to conditional simulations and, consequently, permits now to simulate easily several natural scenes (clouds, mountains) with valleys and fixed topographic points. Such a simulation is presented in Figure 6.

Bibliography

A. Benassi, S. Jaffard, and D. Roux. Elliptic Gaussian random processes. *Revista Mathematica Iberoamericana*, 18:19–89, 1997. [p39]

A. Bonami and A. Estrade. Anisotropic analysis of some Gaussian models. *Journal of Fourier Analysis and Applications*, 9(3):215–236, 2003. [p40]

A. Brouste and S. Lambert-Lacroix. *FieldSim: Random Fields (and Bridges) Simulations*, 2015. URL https://CRAN.R-project.org/package=FieldSim. R package version 3.2.1. [p38]

A. Brouste, J. Istas, and S. Lambert-Lacroix. On fractional Gaussian random fields simulations. *Journal of Statistical Software*, 23(1):1–23, 2007. doi: 10.18637/jss.v023.i01. [p38, 39, 42]

A. Brouste, J. Istas, and S. Lambert-Lacroix. On simulation of manifold indexed fractional Gaussian fields. *Journal of Statistical Software*, 34(4):1–14, 2010. doi: 10.18637/jss.v034.i04. [p38, 39, 40, 42]

name of the process	parameter
	On $[0,1]$
"fBm-line"	`numeric`
"mBm-line"	`function`
"2pfBm-line"	`list(H = numeric, K = numeric)`
"stdfBm-line"	`list(H = numeric, sigma = function, tau = function)`
"cond-fBm-line"	`list(Gamma = matrix, par = numeric)`
"cond-mBm-line"	`list(Gamma = matrix, par = function)`
"cond-2pfBm-line"	`list(Gamma = matrix, par = list(H = numeric, K = numeric))`
	On the square $[0,1]^2$
"fBm-plane"	`numeric`
"mBm-plane"	`function`
"2pfBm-plane"	`list(H = numeric, K = numeric)`
"afBf-plane"	`list(H = numeric, theta1 = numeric, theta2 = numeric)`
"fBs-plane"	`vector`
"cond-fBm-plane"	`list(Gamma = matrix, par = numeric)`
"cond-mBm-line"	`list(Gamma = matrix, par = function)`
"cond-afBf-plane"	`list(Gamma = matrix, par = list(H = numeric, theta1 = numeric, theta2 = numeric))`
"cond-fBs-line"	`list(Gamma = matrix, par = vector)`
	On the hyperboloid
"fBm-hyperboloid"	`numeric`
"cond-fBm-hyperboloid"	`list(Gamma = matrix, par = numeric)`

Table 1: *The "process" class*. `"fBm"` for fractional Brownian motion, `"mBm"` for multifractional Brownian motion, `"2pfBm"` for the standard bi-fractional Brownian motion, `"stdfBm"` for the space-time deformed fractional Brownian motion, `"AfBf"` for anisotropic fractional Brownian field and `"fBs"` for fractional Brownian sheet.

J. Coeurjolly. Simulation and identification of the fractional Brownian motion: A bibliographical and comparative study. *Journal of Statistical Software*, 5(7):1–53, 2000. doi: 10.18637/jss.v005.i07. [p38]

Z. Gelbaum and M. Titus. Simulation of fractional Brownian surfaces via spectral synthesis on manifolds. *IEEE Transactions on Image Processing*, 23(10):4383–4388, 2014. [p38]

A. Kamont. On the fractional anisotropic Wiener fields. *Journal of Probability and Mathematical Statistics*, 18:85–98, 1996. [p40]

A. Kolmogorov. Wienersche Spiralen und einige andere interessante Kurven im Hilbertschen Raum. (German). *C. R. (Doklady) Academy of Sciences URSS*, 26:115–118, 1940. [p38]

B. Mandelbrot and J. Van Ness. Fractional Brownian motions, fractional noises and application. *SIAM Review*, 10:422–437, 1968. [p38]

H. Peitgen and D. Saupe. *The Science of Fractal Images*. Springer-Verlag, 1988. [p38]

R. Peltier and J. Levy-Véhel. Multifractional Brownian motion: Definition and preliminary results. Technical Report RR 2645, INRIA, 1996. URL http://hal.inria.fr/inria-00074045/fr/. [p39]

V. Piterbag. *Asymptotic Methods in the Theory of Gaussian Processes and Fields*. American Mathematical Society, 1996. [p41]

G. Samorodnitsky and M. Taqqu. *Stable non-Gaussian Random Processes: Stochastic Models with Infinite Variance*. Chapman & Hall, New York, 1994. [p39]

M. Schlather, A. Malinowski, P. Menck, M. Oesting, and K. Strokorb. Analysis, simulation and prediction of multivariate random fields with package RandomFields. *Journal of Statistical Software*, 63(8):1–25, 2015. doi: 10.18637/jss.v063.i08. [p38]

M. Schlather, A. Malinowski, M. Oesting, D. Boecker, K. Strokorb, S. Engelke, J. Martini, F. Ballani, O. Moreva, P. J. Menck, S. Gross, U. Ober, Christoph Berreth, K. Burmeister, J. Manitz, O. Morena, P. Ribeiro, R. Singleton, B. Pfaff, and R Core Team. *RandomFields: Simulation and Analysis of Random Fields*, 2016. URL http://CRAN.R-project.org/package=RandomFields. R package version 3.1.8. [p38]

M. Stein. Fast and exact simulation of fractional Brownian surfaces. *Journal of Computational and Graphical Statistics*, 11(3):587–599, 2002. [p38]

A. Wood and G. Chan. Simulation of stationary Gaussian processes in $[0, 1]^d$. *Journal of Computational and Graphical Statistics*, 3(4):409–432, 1994. [p38]

Z. Yin. New methods for simulation of fraction Brownian motion. *Journal of Computational Physics*, 127: 66–72, 1996. [p38]

Alexandre Brouste
Laboratoire Manceau de Mathématiques
Institut du Risque et de l'Assurance du Mans
Université du Maine
72000 Le Mans, France
Alexandre.Brouste@univ-lemans.fr

Jacques Istas
Laboratoire Jean Kuntzmann
Université de Grenoble
38041 Grenoble Cedex 9, France
jacques.istas@upmf-grenoble.fr

Sophie Lambert-Lacroix
UPMF Laboratoire TIMC
Faculté de Médecine
Université de Grenoble
38706 La Tronche Cedex, France
Sophie.Lambert@imag.fr

rTableICC: An R Package for Random Generation of 2×2×K and R×C Contingency Tables

by Haydar Demirhan

Abstract In this paper, we describe the R package **rTableICC** that provides an interface for random generation of 2×2×K and R×C contingency tables constructed over either intraclass-correlated or uncorrelated individuals. Intraclass correlations arise in studies where sampling units include more than one individual and these individuals are correlated. The package implements random generation of contingency tables over individuals with or without intraclass correlations under various sampling plans. The package include two functions for the generation of K 2×2 tables over product-multinomial sampling schemes and that of 2×2×K tables under Poisson or multinomial sampling plans. It also contains two functions that generate R×C tables under product-multinomial, multinomial or Poisson sampling plans with or without intraclass correlations. The package also includes a function for random number generation from a given probability distribution. In addition to the contingency table format, the package also provides raw data required for further estimation purposes.

Introduction

Random generation of contingency tables is essential for simulation studies conducted over categorical data. The main characteristic of a contingency table is determined by the assumed sampling plan and the correlation structure between categorical variables constituting the table. There are three main sampling plans: Poisson, multinomial, and product multinomial. In the Poisson plan, each cell is independently Poisson distributed and there is no restriction on the total sample size. In the multinomial plan, total sample size is fixed while row and column totals are not fixed. When one of the margins of the table is fixed and the rest are set free, we have a product multinomial plan (Agresti, 2002; Bishop et al., 1975). If both margins are naturally fixed, the sampling plan becomes hypergeometric, which is seldom used in practice (Agresti, 2002). There are numerous ways in R to generate contingency tables of various dimensions. The function `r2dtable()` in the base package **stats** generates random two-way tables with given marginals using Patefield's algorithm under product-multinomial sampling (Patefield, 1981). Alternatively, one can generate a random contingency table over log-linear models with a predetermined association structure. However, there is no package in R for random generation of 2×2×K tables or generation of contingency tables with intraclass-correlations.

It is highly possible to have intraclass correlations (ICCs) in surveys conducted over sampling units with more than one observation unit if these units are correlated. Familial data also include ICCs. In a public health survey, if data are collected over families, intraclass correlations arise due to the within family dependence. Presence of intraclass correlations can invalidate results of classical categorical models or chi-square tests (Demirhan, 2013). Therefore, use and further developments of methods specific to the cases with ICCs are essential. In the literature, Cohen (1976) and Altham (1976) introduced categorical analyzes under the presence of ICCs. Borkowf (2000) proposed an ICC statistic for contingency tables with the empirical multivariate quantile-partitioned distributions. Nandram and Choi (2006) proposed Bayesian analysis of R×C tables with intraclass correlated cells. Demirhan (2013) proposed Bayesian estimation of log odds ratios over R×C contingency tables under the presence of intraclass correlated cells. The context of ICCs is also used in applied research such as Bi and Kuesten (2012).

Monte Carlo simulation studies are essential in the development of new statistical methods to handle ICCs. However, there is neither a Monte Carlo approach nor an R package to implement random generation of contingency tables under intraclass-correlated individuals. In this article, we propose a simple approach for the generation of 2×2×K and R×C contingency tables in the presence of ICCs between individuals under three sampling plans, and describe the R package **rTableICC** (Demirhan, 2015) for the implementation of the proposed approach. In general, 2×2×K tables are observed in multicenter studies such as clinical trials (Demirhan and Hamurkaroglu, 2008). Also, in a genetic association study, association between existence of a disease and *K* single-nucleotide polymorphisms (SNPs) can be questioned over a 2 × 2×K contingency table. In the genetics context, K would be the number of genetic loci under investigation. The assumption is that the total sample size under each loci is mostly known. It is highly possible to have some correlation patterns between SNPs that cause existence of ICCs. Thus, we have a 2 × 2×K table over individuals with ICCs under product-multinomial sampling plan. R×C tables provide a general framework for two-way contingency tables.

Considering the areas of application, **rTableICC** provides a rich platform for the random generation of contingency tables.

The package **rTableICC** includes four functions for random generation of 2×2×K and R×C contingency tables with and without intraclass-correlated individuals under multinomial, product - multinomial and Poisson sampling plans. It also has a function for random generation of data from a given probability function. Generated tables are made available in both table and raw data format. Additional characteristics of generated data for further estimation issues are also produced and optionally printed out. Thus, it is possible to easily embed functions of **rTableICC** in other Monte Carlo simulation codes. The latest development of **rTableICC** under version 1.0.3 is published on the Comprehensive R Archive Network (CRAN).

In the following sections, the approach for the generation of random tables in the presence of ICCs is described, details of data generation processes under considered sampling plans are mentioned, input and output structures of **rTableICC** are demonstrated, and use of the package is illustrated by several examples. We also provide a performance analysis regarding the mean running times of the functions in the package **rTableICC**. Then, we conclude with a brief summary.

Data generation under ICC

Altham (1976) introduced two probabilities to deal with ICCs over an R×C contingency table. Let n_{ijk} be the number of individuals falling in the cell (j,k) of an R×C table from the ith cluster, where $i = 1,\ldots,I$, $j = 1,\ldots,R$, $k = 1,\ldots,C$, and π_{jk} be the related cell probability. The total number of individuals in the ith cluster is shown by n_i and the intraclass correlation coefficient for clusters including $t = n_i$ individuals is denoted by θ_t for $t = 2,\ldots,T$, where T is the greatest family size and $\theta_1 = 0$. For the events $A = \{$All individuals in the ith cluster fall in the same cell of an R×C table $\}$ and $B = \{$Individuals are in different but specified cells$\}$, the following probabilities are given by Altham (1976):

$$\mathbb{P}(A) = \theta_t \pi_{jk} + (1-\theta_t)(\pi_{jk})^t \tag{1}$$

and

$$\mathbb{P}(B) = (1-\theta_t)\prod_{j=1}^{R}\prod_{k=1}^{C}(\pi_{jk})^{n_{ijk}}, \tag{2}$$

where $0 \leq \theta_t \leq 1$. For $2 \times 2 \times$K tables, equations (1) and (2) remain the same but $i, j = 1, 2$.

We utilize equations (1) and (2) to incorporate ICCs into the data generation process. We work over clusters to generate data. For all sampling plans, the total sample size either entered or obtained over randomly generated data is distributed across the clusters. Then, for the clusters with only one individual, because there is no ICC affecting the individual, we randomly assign it to one of the cells of the table taking the input vector of cell probabilities into account, $\boldsymbol{\pi}$. For clusters with more than one individual, we employ the following pseudocode algorithm to generate data under the given ICCs:

Algorithm 1.

1. Input $\boldsymbol{\theta}$, $\boldsymbol{\pi}$, and number of individuals in each cluster by an $M \times 1$ vector $\boldsymbol{m}$;
2. Set $i = 1$ and goto step 3;
3. Generate all possible compositions of order $R \times C$ of cluster size m_i into at most m_i parts;
4. Write generated compositions to an $r \times \ell$ matrix $\boldsymbol{N}$, where r is the total number of possible compositions;
5. For each composition $\boldsymbol{n_j}$, if $\sum_k n_{jk} = 0$, compute the probability p_j by equation (1), else if $\sum_k n_{jk} > 0$, compute the probability p_j by equation (2), for $j = 1,\ldots,r$;
6. Normalize the series of probabilities, $\boldsymbol{p}$, obtained at step 5 to construct a probability function;
7. Randomly select one of the compositions based on the probability function obtained at step 6.
8. Write selected composition to an $\ell \times 1$ vector $\boldsymbol{s}_i$ and set $i = i + 1$;
9. If $i \leq M$ goto step 3, else return $\sum_i \boldsymbol{s}_i$.

In Algorithm 1, $\ell = R \cdot C$ for R×C tables and $\ell = 4$ for $2 \times 2 \times$K tables. We use the function `compositions` from the package **partitions** (Hankin, 2006) to generate all possible compositions at the step 3 of Algorithm 1. Each composition represents one of the possible allocations of individuals in a cluster into target cells. For example, let us have 4 cells to distribute 5 individuals in a cluster. We run the following code to get the 56×4 matrix $\boldsymbol{N}$:

```
> N <- t(compositions(5, 4, include.zero = TRUE))
```

The resulting output looks like

```
[1,] 5 0 0 0   [2,] 4 1 0 0   [3,] 3 2 0 0   [4,] 2 3 0 0 ...
```

The vector $(5,0,0,0)$ implies that all individuals in the cluster of interest fall in the first (same) cell and corresponds to the event A, whereas the vector $(2,3,0,0)$ implies that 2 of 5 individuals fall in the first and the rest fall in the second cell and represents the event B. At the step 6 of Algorithm 1, we normalize the set of probabilities that consists of the probability of each possible allocation of individuals in the cluster of interest into the cells of table. By this way, we form a probability distribution to generate one of the possible allocation randomly. Consequently, individuals in a cluster of size more than one are distributed into the cells of the table by Algorithm 1. After application of Algorithm 1 for all clusters, the grand total of generated cell counts produces a randomly generated contingency table.

Structure of the rTableICC package

The package **rTableICC** consists of four main functions: `rTableICC.RxC`, `rTableICC.2x2xK`, `rTable.RxC` and `rTable.2x2xK`; and an auxiliary function `rDiscrete`, which is also suitable for use individually. In the general functioning of the package, first, main inputs are checked by an initial layer according to the presence of ICCs and used sampling plan; and then the related function is called. In addition to general checks, specific checks are done by the related function itself. Below, we describe the processing of each function after the general check.

Generation of R×C tables with ICC

The function `rTableICC.RxC` is called to generate an R×C table with ICC. Algorithm 2 describes the functioning of `rTableICC.RxC`.

Algorithm 2.

1. Input sampling plan, $\boldsymbol{\theta}$, $\boldsymbol{\pi}$, total number of individuals N or mean number of individuals $\boldsymbol{\lambda}$, and total number of clusters M;
2. If sampling plan is multinomial goto step 3, product-multinomial goto step 7, and Poisson goto step 15;
3. If any of inputs $\boldsymbol{\pi}$ and total number of individuals is not suitable then stop;
4. Distribute N individuals across M clusters with equal probabilities by `rmultinom(1,N,rep(1/M, M))`;
5. If the maximum number of individuals in one of the clusters is greater than the maximum allowed cluster size then stop;
6. Employ Algorithm 1 with joint probabilities for all clusters and goto step 21;
7. If any of inputs $\boldsymbol{\pi}$ and row (column) margins is not suitable then stop;
8. Determine the fixed margin according to input parameters `col.margin` or `row.margin` and set $i = 1$;
9. Calculate conditional probabilities regarding the fixed margin;
10. If conditional probabilities calculated over entered row margins and $\boldsymbol{\pi}$ are not equal to each other then stop;
11. Distribute individuals in the ith row (column) across M clusters with equal probabilities by using the multinomial distribution;
12. If the maximum number of individuals in one of the clusters is greater than the maximum allowed cluster size then stop;
13. Employ Algorithm 1 with calculated conditional probabilities for all clusters and set $i = i + 1$;
14. If $i \leq R(C)$ goto step 10, else goto step 21;
15. If input $\boldsymbol{\lambda}$ is not suitable then stop;
16. Generate number of individuals in each cell by `rpois(R * C,t(lambda))`;
17. Calculate cell probabilities and total number of individuals N;
18. Distribute N individuals across M clusters with equal probabilities by `rmultinom(1,N,rep(1/M, M))`;

19. If the maximum number of individuals in one of the clusters is greater than the maximum allowed cluster size then stop, else goto step 20;
20. Employ Algorithm 1 with probabilities calculated at step 17 for all clusters;
21. Calculate desired output forms of generated table.

Suitability checks at steps 3, 7, and 15 are made on minimum and maximum values and dimensions of input vectors. Because the total sample size, which is entered by the user for multinomial sampling, randomly generated for Poisson sampling, and entered as a fixed row (column) margin for product-multinomial sampling, is randomly distributed into the clusters, it is coincidentally possible to have clusters with more individuals than the allowed maximum cluster size. In this case, the following error message is generated:

```
Maximum number of individuals in one of the clusters is 14,which is greater than maximum
allowed cluster size. (1) Re-run the function,(2) increase maximum allowed cluster size
by increasing the number of elements of theta,(3) increase total number of clusters,or
(4) decrease total number of individuals!
```

and execution is stopped at steps 5, 12, and 19 of Algorithm 2.

For the product-multinomial sampling, suppose that row totals are fixed and n_{i+} denotes fixed row margins. With the counts satisfying $\sum_j n_{ij} = n_{i+}$, we have the following multinomial form (Agresti, 2002):

$$\frac{n_{i+}!}{\prod_j n_{ij}!} \prod_j \pi_{j|i}^{n_{ij}}, \tag{3}$$

where $i = 1, \ldots, R$, $j = 1, \ldots, C$, n_{ij} is the count of cell (i, j), and given that an individual is in the ith row, $\pi_{j|i}$ is the conditional probability of being in the jth column of the table calculated at step 9 of Algorithm 2. When column totals are fixed the same steps as in the case of fixed row totals are applied.

Let Λ be the set of clusters in which all individuals fall in a single cell of the contingency table and Λ' be the complement of Λ, and T be the maximum cluster size. Outputs of `rTableICC.RxC` include two arrays in addition to the generated table. The first one, g_t, is an $R \times C \times (T-1)$ dimensional array including the number of clusters of size t in Λ' with all individuals in cell (i, j); and the second, $\tilde{g}$, is a $(T-1) \times 1$ dimensional vector including the number of clusters of size t in Λ', where $i, j = 1, 2$ and $t = 2, \ldots, T$. These arrays are required for further modeling purposes.

Generation of $2 \times 2 \times$K tables with ICC

The function `rTableICC.2x2xK` is called to generate a $2 \times 2 \times$K table with ICC. Algorithm 3 describes the processing of `rTableICC.2x2xK`. We assume that we have K centers and a 2×2 table under each center. To generate a $2 \times 2 \times$K table, `rTableICC.2x2xK` generates a 2×2 table under each center.

Algorithm 3.

1. Input sampling plan, $\boldsymbol{\theta}$, $\boldsymbol{\pi}$, total number of individuals N or mean number of individuals $\boldsymbol{\lambda}$, and total number of clusters M_k for $k = 1, \ldots, K$ under each center;
2. If sampling plan is multinomial goto step 3, product-multinomial goto step 9, and Poisson goto step 16;
3. If any of inputs $\boldsymbol{\pi}$ and total number of individuals is not suitable then stop;
4. Distribute N individuals across $\sum_k M_k$ clusters with equal probabilities by `rmultinom(1,N, rep(1/sum(num.cluster),sum(num.cluster)))` and store the results in a $K \times 1$ vector $\boldsymbol{c}$;
5. If the maximum number of individuals in one of the clusters is greater than the maximum allowed cluster size then stop, else set $k = 1$;
6. Scale joint probabilities of the 2×2 table under the kth center to make them sum-up to one;
7. Employ Algorithm 1 with scaled joint probabilities for all clusters of center k and set $k = k + 1$;
8. If $k \leq K$ goto step 6, else goto step 22;
9. If any of inputs $\boldsymbol{\pi}$ and center margins is not suitable then stop;
10. Calculate conditional probabilities regarding the fixed centers and set $k = 1$;
11. Scale conditional probabilities of step 10 under the kth center to make them sum-up to one;
12. Distribute individuals in the kth center across M_k clusters with equal probabilities by `rmultinom(1, N[k],rep(1/num.cluster[k],num.cluster[k]))`;

13. If the maximum number of individuals in one of the clusters is greater than the maximum allowed cluster size then stop;
14. Employ Algorithm 1 with scaled conditional probabilities for all clusters of center k and set $k = k + 1$;
15. If $k \leq K$ goto step 11, else goto step 22;
16. If input $\boldsymbol{\lambda}$ is not suitable then stop;
17. Generate number of individuals in each cluster by `rpois(num.cluster[k],lambda[k])`;
18. Calculate total number of individuals N over generated clusters at step 17;
19. Scale joint probabilities of the 2×2 table under the kth center to make them sum-up to one;
20. If the maximum number of individuals in one of the clusters is greater than the maximum allowed cluster size then stop;
21. Employ Algorithm 1 with probabilities calculated at step 19 for all clusters;
22. Calculate desired output forms of generated table.

Suitability checks at steps 3, 9, and 16 are made on minimum and maximum values and dimensions of input vectors. For the incompatibility between generated and allowed maximum cluster sizes, the same situation as the R×C case also applies to the 2 × 2×K case. In this case, the same error message is displayed and execution is stopped. For all sampling plans, `rTableICC.2x2xK` proceeds over each center.

For product-multinomial sampling plan, suppose that center totals are denoted by n_{ij+}, where $i, j = 1, 2$. Then with the counts satisfying $\sum_{ij} n_{ijk} = n_{ij+}$, the following multinomial form is used (Agresti, 2002):

$$\frac{n_{ij+}!}{\prod_{ij} n_{ijk}!} \prod_{ij} p_{ij|k}^{n_{ijk}}, \tag{4}$$

where $k = 1, \ldots, K$, n_{ijk} is the count of cell (i, j, k), and given that an individual is in the kth center, $p_{ij|k}$ is the conditional probability of being in the cell (i, j) of the 2×2 table. This multinomial form is used to generate data under each center.

Arrays $\boldsymbol{g}_t$ and $\widetilde{\boldsymbol{g}}$ are also included in the outputs of `rTableICC.2x2xK`. Here, $\boldsymbol{g}_t$ and $\widetilde{\boldsymbol{g}}$ are respectively $2K \times 2 \times (T-1)$ and $(T-1) \times 1$ dimensional arrays. Their definitions are the same as R×C case.

Generation of R×C tables without ICC

The function `rTable.RxC` is used to generate an R×C table with independent individuals in sampling units. In this function, the classical way of generating contingency tables over the probability distribution corresponding to the sampling plan is followed. The functioning of `rTable.RxC` is described in Algorithm 4.

Algorithm 4.

1. Input sampling plan, $\boldsymbol{\pi}$, and total number of individuals N or mean number of individuals $\boldsymbol{\lambda}$;
2. If sampling plan is multinomial goto step 3, product-multinomial goto step 5, and Poisson goto step 11;
3. If any of inputs $\boldsymbol{\pi}$ and total (mean) number of individuals is not suitable then stop;
4. Distribute N individuals across R×C cells by `rmultinom(1,N,pi)` and goto step 12;
5. If any of inputs $\boldsymbol{\pi}$ and row (column) margins is not suitable then stop;
6. Determine the fixed margin according to input parameters `col.margin` or `row.margin` and set $i = 1$;
7. Calculate conditional probabilities regarding the fixed margin;
8. If conditional probabilities calculated over entered row margins and $\boldsymbol{\pi}$ are not equal to each other then stop;
9. Distribute individuals in the ith row (column) across R (C) cells with conditional probabilities using the multinomial distribution;
10. If $i \leq R(C)$ goto step 9, else goto step 13;
11. If input $\boldsymbol{\lambda}$ is not suitable then stop;
12. Generate number of individuals in each cell by `rpois(R * C,t(lambda))`;
13. Calculate desired output forms of generated table.

Suitability checks at steps 3, 5, and 11 are made on minimum and maximum values and dimensions of input vectors. For the product-multinomial sampling plan, the multinomial form in equation (3) is used. Raw data corresponding to each individual are also generated among outputs of `rTable.RxC`.

Generation of $2 \times 2 \times$K tables without ICC

The function `rTable.2x2xK` is employed to generate a $2 \times 2 \times$K table with independent individuals in sampling units. The processing of `rTable.2x2xK` is described in Algorithm 5. Assume that we have K centers and a 2×2 table under each center. Similar to `rTableICC.2x2xK`, `rTable.2x2xK` generates a 2×2 table under each center to obtain a $2 \times 2 \times$K table.

Algorithm 5.

1. Input sampling plan, $\boldsymbol{\pi}$, total number of individuals N or mean number of individuals $\boldsymbol{\lambda}$;
2. If sampling plan is multinomial goto step 3, product-multinomial goto step 5, and Poisson goto step 10;
3. If any of inputs $\boldsymbol{\pi}$ and total number of individuals is not suitable then stop;
4. Distribute N individuals across $2 \times 2 \times$ K cells with input probabilities by `rmultinom(1,N,pi)` and goto step 12;
5. If any of inputs $\boldsymbol{\pi}$ and center margins is not suitable then stop, else set $k = 1$;
6. Calculate conditional probabilities for center k;
7. Scale conditional probabilities of step 6 under the kth center to make them sum-up to one;
8. Distribute individuals in the kth center across 2×2 cells with scaled probabilities at step 7 by using multinomial distribution and set $k = k + 1$;
9. If $k \leq K$ goto step 6, else goto step 12;
10. If input $\boldsymbol{\lambda}$ is not suitable then stop;
11. Generate number of individuals in each cell of $2 \times 2 \times$K table by `rpois(2 * 2 * K,lambda)`;
12. Calculate desired output forms of generated table.

Suitability checks at steps 3, 9, and 16 are made on minimum and maximum values and dimensions of input vectors.The multinomial form in equation (4) is used for product-multinomial sampling plan. It is possible to enter a mean number of individuals for each cell under Poisson sampling plan at step 11 of Algorithm 5 by entering an array for `lambda`. Raw data corresponding to each individual are also generated among outputs of `rTable.2X2XK`.

Generation of random values from a discrete probability distribution

The function `rDiscrete` is used to generate a random value from an empirical probability distribution. This function is called by both `rTableICC.RxC` and `rTableICC.2x2xK`. Implementation of `rDiscrete` is explained by Algorithm 6.

Algorithm 6.

1. Input empirical probability function (pf) with N levels and number of observations to be generated;
2. Check whether input probabilities sum to one and number of observations n is a finite positive scalar;
3. Calculate cumulative distribution function (cdf), F, over the input pf;
4. Set $A_j = (F(j-1), F(j))$, where $j = 1, \ldots, N$, $F(0) = 0$, and $i = 1$;
5. Generate a random value u from Uniform$(0, 1)$ distribution;
6. If $u \in A_j$ than save j as the generated value and set $i = i + 1$;
7. If $i \leq n$ goto step 5;
8. Return the generated values.

`rDiscrete` returns an array of generated values and calculated cdf at step 3 of Algorithm 6.

Illustrative examples

To generate random R$\times$C and $2 \times 2 \times$K contingency tables with or without ICCs or generate random numbers from empirical probability functions, first one has to load the package **rTableICC** by

```
> library(rTableICC)
```

Then, the relevant function is called with proper inputs.

In the first example, we illustrate two important cases that generate errors and stop execution of functions `rTableICC.RxC` and `rTableICC.2x2xK`. In the second and third examples, we demonstrate outputs of `rTableICC.2x2xK` and `rTableICC.RxC`. In the fourth example, we exemplify `rTable.RxC`, `rTable.2x2xK`, and `rDiscrete` functions.

Example 1

In this example, we illustrate two incompatibilities between generated and allowed maximum cluster sizes and total number of individuals and number of clusters for functions `rTableICC.RxC` and `rTableICC.2x2xK`.

When a user enters the value of intraclass correlation for each cluster size, the maximum allowed cluster size is correspondingly defined. However, because `rTableICC.RxC` and `rTableICC.2x2xK` distribute total sample size, which is entered or generated, among the given number of clusters, we would have clusters with number of individuals greater than the maximum allowed cluster size. This case should be regarded while entering the values of intraclass correlations, total or mean number of individuals, and total number of clusters.

The following code attempts to generate a $2 \times 2 \times$K contingency table with 3 centers under multinomial sampling plan. Number of clusters under each sample is 25 and total number of individuals is 500. The maximum cluster size (`max.cluster.size`) is defined to specify the size of array including ICCs. In this setting, it is highly possible to allocate more than 4 individuals in one of the clusters.

```
> num.centers <- 3
> sampl <- "Multinomial"
> max.cluster.size <- 4
> num.cluster <- 25
> num.obs <- 500
> ICCs <- array(0.1, dim = max.cluster.size)
> ICCs[1] <- 0
> cell.prob <- array(1/12, dim = c(num.centers, 4))
> x <- rTableICC.2x2xK(p = cell.prob, theta = ICCs, M = num.cluster, sampling = sampl,
+                      N = num.obs)
```

When 500 individuals are distributed across 25 clusters, the maximum cluster size is realized as $14 >$ `max.cluster.size`, as expected. Then, execution is stopped with the following error message:

```
Error in rtableICC.2x2xK.main(p, theta, M, sampling, N, lambda, print.regular,  :
        Maximum number of individuals in one of the clusters is 14, which is greater
        than maximum allowed cluster size.
           (1) Re-run the function,
           (2) increase maximum allowed cluster size by increasing the number of
               elements of theta,
           (3) increase total number of clusters, or
           (4) decrease total number of individuals!
```

Now, we change the settings to eliminate the error. `rTableICC.2x2xK` generates the desired table when the total number of observations is decreased to 50, the total number of clusters is increased to 250, or the maximum cluster size is increased to 15 with the same inputs for the rest of the arguments.

User should ensure compatibility between the number of individuals and the total number of clusters. When we run the code given above with `num.obs <-50` and `zero.clusters <-FALSE`, `rTableICC.2x2xK` tries to distribute 50 individuals to 75 clusters; and hence, the following error message is generated:

```
Error in rtableICC.2x2xK.main(p, theta, M, sampling, N, lambda, zero.clusters,  :
  Because number of individuals is less than the total number of clusters, it is
  impossible to allocate an individual to each cluster! Set zero.clusters = TRUE
  and re-run the function.
```

The problem is eliminated when `zero.clusters` is set to `TRUE`.

Example 2

In this example, the output structure of `rTableICC.2x2xK` is illustrated. We run the code in Example 1 with `num.centers <-2`, `num.obs <-50`, and `zero.clusters <-TRUE` and call `print(x)`. The following part presents the summary information on the data generation process.

```
Call:
rTableICC.2x2xK.default(p = cell.prob, theta = ICCs, M = num.cluster,
    sampling = sampl, N = num.obs, zero.clusters = TRUE, print.regular = TRUE,
    print.raw = FALSE)

    Process summary:
    ----------------
    100 observations in 2 centers were successfully generated under Multinomial
    sampling plan! Number of clusters for each center is as the following:
      25 for Center 1
      25 for Center 2
    17 clusters include no individual.
    21 clusters include one individual.
    12 clusters include more than one individual.
```

Because the multinomial distribution is used to distribute the total sample size across the clusters, there are some clusters with no individuals, as reported in the process summary. Because probabilities used to represent intraclass correlations in equations (1) and (2) change according to cluster size, we report the number of clusters containing one and more than one individuals in the process summary.

The following part of the output includes g_t, $\widetilde{g}$, and the generated table in two and three dimensions.

```
 The number of t sized clusters in the set of clusters in which all individuals fall
 in cell (j,k) for j,k=1,2:

   g.t =
, , Cluster of size 2
               C- 1 C- 2
Center- 1 R- 1    0    2
Center- 1 R- 2    1    2
Center- 2 R- 1    1    0
Center- 2 R- 2    0    1
, , Cluster of size 3
               C- 1 C- 2
Center- 1 R- 1    1    1
Center- 1 R- 2    0    0
Center- 2 R- 1    0    1
Center- 2 R- 2    0    0
, , Cluster of size 4
               C- 1 C- 2
Center- 1 R- 1    0    0
Center- 1 R- 2    0    0
Center- 2 R- 1    0    0
Center- 2 R- 2    0    1

 The number of clusters of size t outside the set of clusters in which all individuals
 fall in a single cell: g.tilde = ( 0 1 0 )

 Generated random table in two dimensions :
          R1C1 R1C2 R2C1 R2C2
Center- 1    4   10    7    7
Center- 2    3    5    5    9

   Generated random table in three dimensions :
, , Center- 1
     C- 1 C- 2
R- 1    4   10
R- 2    7    7
, , Center- 2
     C- 1 C- 2
R- 1    3    5
R- 2    5    9
```

To illustrate the output raw data format, we run the following code:

```
> num.centers <- 3
```

```
> num.cluster <- 5
> num.obs <- 10
> ICCs <- array(0.1, dim = 4)
> ICCs[1] <- 0
> cell.prob <- array(1/12, dim = c(num.centers, 4))
> x <- rTableICC.2x2xK(p = cell.prob, theta = ICCs, M = num.cluster,
+                      sampling = "Multinomial", N = num.obs)
```

The resulting raw data output given below is printed as a three dimensional array. The first dimension includes observations, the second dimension has $2K$ elements simultaneously representing rows of each 2×2 table and each center, and the third dimension corresponds to the columns of each 2×2 table. Elements of the second dimension correspond to cells in (row-1, center-i), (row-2, center-i), for $i = 1, \ldots, K$, respectively; hence, it has $2K$ elements. Those of the third dimension correspond to the first and second columns of each 2×2 table, respectively.

```
   Generated random table in raw data format =
   , , 1
      [,1] [,2] [,3] [,4] [,5] [,6]
 [1,]    1    0    0    0    0    0
 [2,]    0    0    0    0    0    0
 [3,]    1    0    0    0    0    0
 ...
 [10,]   0    0    0    0    0    0
  , , 2
      [,1] [,2] [,3] [,4] [,5] [,6]
 [1,]    0    0    0    0    0    0
 [2,]    1    0    0    0    0    0
 [3,]    0    0    0    0    0    0
 ...
 [10,]   0    0    0    0    1    0
```

Example 3

The output structure of `rTableICC.RxC` is similar to that of `rTableICC.2x2xK`. We run the following code to generate a 2×3 contingency table under a product multinomial sampling plan with fixed row margins, zero clusters being not allowed, and cell probabilities being in accordance with the entered counts of fixed margin.

```
> num.cluster <- 12
> ICCs <- array(0.1, dim = 9)
> ICCs[1] <- 0
> num.obs <- 24
> zeros <- FALSE
> sampl <- "Product"
> row <- c(12, 12)
> cell.prob <- array(0, dim = c(2, 3))
> cell.prob[1, 1:2] <- 0.2
> cell.prob[1, 3] <- 0.1
> cell.prob[2, 1:2] <- 0.1
> cell.prob[2, 3] <- 0.3
> y <- rTableICC.RxC(p = cell.prob, theta = ICCs, row.margins = row, M = num.cluster,
+                    sampling = sampl, zero.clusters = zeros, print.regular = TRUE,
+                    print.raw = FALSE)
> print(y)
```

In the output of `rTableICC.RxC`, first the following summary table is generated. Coincidentally, there is no cluster with more than one individual. Clusters are enforced to contain at least one individual.

```
Call:
rTableICC.RxC.default(p = cell.prob, theta = ICCs, M = num.cluster,
    row.margins = row, sampling = sampl, zero.clusters = zeros,
    print.regular = TRUE, print.raw = FALSE)

    Process summary:
    ----------------
    24 observations in 12 12 clusters were successfully generated under Product
```

```
    multinomial sampling plan!
    Each cluster includes at least one individual.
    12 clusters include one individual.
    0 clusters include more than one individual.
```

In the output, the vector g_t is printed in $R \times C$ format for each cluster size. The vector $\widetilde{g}$ is printed as a vector and the generated table is printed in both $R \times C$ and row formats. Because there is no cluster with more than one individual, g_t and $\widetilde{g}$ are both composed of zeros.

```
 The number of t sized clusters in the set of clusters in which all individuals fall in
 cell (j,k) for j=1,...,R and k=1,...,C: g.t =

, , Cluster of size 2
     C- 1 C- 2 C- 3
R- 1    0    0    0
R- 2    0    0    0
...
, , Cluster of size 9
     C- 1 C- 2 C- 3
R- 1    0    0    0
R- 2    0    0    0

 The number of clusters of size t outside the set of clusters in which all individuals
 fall in a single cell: g.tilde = ( 0 0 0 0 0 0 0 0 )

 Generated random table in row format = ( 5 4 3 3 3 6 )

   Generated random table in RxC format =
     C- 1 C- 2 C- 3
R- 1    5    4    3
R- 2    3    3    6
```

Example 4

In this example, we run a couple of codes to illustrate random contingency table generation without ICCs. Besides, we show outputs of the function `rDiscrete`.

The following code generates and prints a random 5×7 contingency table under multinomial sampling plan with 124 observations and equal cell probabilities.

```
> num.row <- 5
> num.col <- 7
> sampl <- "Multinomial"
> cell.prob <- array(1/35, dim = c(num.row, num.col))
> num.obs <- 124
> x <- rTable.RxC(p = cell.prob, sampling = sampl, N = num.obs)
> print(x)
```

The corresponding output of `rTable.RxC` is as follows. After a brief summary, the generated table is printed.

```
Call:
rTable.RxC.default(p = cell.prob, sampling = sampl, N = num.obs)

    Process summary:
    ----------------
    124 observations across 5 rows and 7 columns were successfully generated under
    Multinomial sampling plan!

   Generated random table in RxC format =
     C- 1 C- 2 C- 3 C- 4 C- 5 C- 6 C- 7
R- 1    4    2    3    4    5    4    4
R- 2    4    5    5    5    5    5    4
R- 3    4    1    5    3    3    2    3
R- 4    2    1    1    6    4    3    3
R- 5    2    1    4    5    7    2    3
```

The following code is run to randomly generate a $2 \times 2 \times 3$ contingency table under Poisson sampling plan with determined mean number of individuals for each cell.

```
> num.centers <- 3
> sampl <- "Poisson"
> cell.mean <- array(3, dim = c(2, 2, num.centers))
> z <- rTable.2x2xK(sampling = sampl, lambda = cell.mean)
> print(z)
```

Consequently, 31 observations were generated under 3 centers.

```
Call:
rTable.2x2xK.default(sampling = sampl, lambda = cell.mean)

    Process summary:
    ----------------
    31 observations in 3 centers were successfully generated under Poisson sampling plan!

   Generated random table in 2x2xK format =

, , Center- 1
     C- 1 C- 2
R- 1    1    0
R- 2    2    4
, , Center- 2
     C- 1 C- 2
R- 1    6    3
R- 2    2    4
, , Center- 3
     C- 1 C- 2
R- 1    2    3
R- 2    2    2
```

To generate random values from an empirical probability function, we call `rDiscrete`. We run the following code to generate two random values from a given probability function:

```
> p <- c(0.23, 0.11, 0.05, 0.03, 0.31, 0.03, 0.22, 0.02)
> rDiscrete(n = 2, pf = p)
```

Consequently, the generated random values and corresponding cdf are printed.

```
$rDiscrete
[1] 1 7

$cdf
[1] 0.00 0.23 0.34 0.39 0.42 0.73 0.76 0.98 1.00
```

Performance

The package **rtableICC** is intended to be used in combinaion with other code that implements Monte Carlo simulation. Therefore, the computational performance of **rtableICC** is of importance. We investigate running times of functions in **rtableICC** under various combinations of table structure, sample size, and sampling plan. Tables 1 and 2 show test conditions of each function of **rtableICC** related with $2 \times 2 \times K$ and $R \times C$ contingency tables, respectively. The value of ICC is taken as 0.1 for all cluster sizes and related functions. Each test combination was repeated 5 times and mean and variance of the running times were recorded. Because of the obtained small variances, 5 replications were found sufficient. The maximum number of allowed clusters was taken high enough to have the code successfully run through. In the `rTableICC.2x2xK` and `rTableICC.RxC` functions, the argument `zero.clusters` was set to `TRUE` to allow clusters with no individuals. Note that when `zero.clusters` is set to `FALSE`, we get shorter mean running times. All the combinations were run on a MAC-Pro computer equipped with 6 Intel(R) Xenon(R) CPU E5-1650 v2 at 3.5GHz, 16 GB of RAM, and Windows 8.1 operating system.

For multinomial, Poisson, and product multinomial sampling plans, scatter plots representing the mean running times of `rTableICC.2x2xK` according to some of the considered factors are given in Figure 1. Due to the small variances within repetitions, plots are drawn only for the mean running times.

	Plan	N. of Observations	N. of Centers	N. of Clusters	Cell Mean	Center Margins
`rTableICC.2x2xK`	Mult.	$10, 25, 50, \ldots, 2 \cdot 10^3$	$2, 4, \ldots, 100$	$5, 6, \ldots, 100$	—	—
`rTable.2x2xK`	Mult.	$10, 25, 50, \ldots, 2 \cdot 10^4$	$2, 4, \ldots, 20$	—	—	—
`rTableICC.2x2xK`	Poi.	—	$2, 4, \ldots, 20$	$5, 10, \ldots, 100$	$1, 2, \ldots, 10$	—
`rTable.2x2xK`	Poi.	—	$2, 4, \ldots, 50$	—	$0.5, 1, \ldots, 50$	—
`rTableICC.2x2xK`	Pro.	—	$2, 4, \ldots, 20$	$5, 10, \ldots, 100$	—	$5, 10, \ldots, 200$
`rTable.2x2xK`	Pro.	—	$2, 4, \ldots, 50$	—	—	$5, 10, \ldots, 500$

N: Number; Mult: Multinomial; Poi: Poisson; Pro: Product Multinomial; Plan: Sampling plan.

Table 1: Test conditions for the `rTableICC.2x2xK` and `rTable.2x2xK` functions.

	Plan	N. of Obs.	N. of Rows	N. of Columns	N. of Clusters	Cell Mean	Row Margins
`rTableICC.RxC`	Mult.	$10, 20, \ldots, 200$	$2, 3, \ldots, 5$	$R, R+1, \ldots, 5$	$5, 10, \ldots, 100$	—	—
`rTable.RxC`	Mult.	$25, 50, \ldots, 10^4$	$2, 3, \ldots, 20$	$R, R+1, \ldots, 20$	—	—	—
`rTableICC.RxC`	Poi.	—	$2, 3, \ldots, 5$	$R, R+1, \ldots, 20$	$15, 20, \ldots, 100$	$1, 2, \ldots, 7$	—
`rTable.RxC`	Poi.	—	$2, 3, \ldots, 20$	$R, R+1, \ldots, 20$	—	$0.5, 1, \ldots, 10$	—
`rTableICC.RxC`	Pro.	—	$2, 3, \ldots, 5$	$R, R+1, \ldots, 5$	$20, 25, \ldots, 40$	—	$20, 25, \ldots, 200$
`rTable.RxC`	Pro.	—	$2, 3, \ldots, 10$	$R, R+1, \ldots, 10$	—	—	$5, 10, \ldots, 2000$

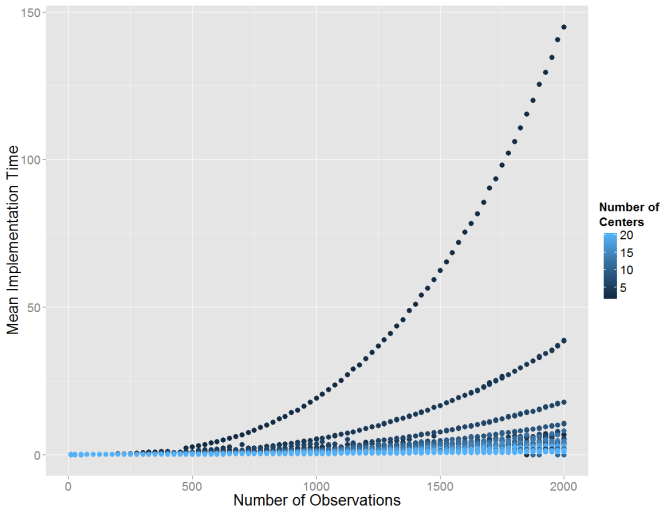
N: Number; Mult: Multinomial; Poi: Poisson; Pro: Product Multinomial; Obs: Observations; Plan: Sampling plan.

Table 2: Test conditions for the `rTableICC.RxC` and `rTable.RxC` functions. The number of columns starts from number of rows denoted by R under number of columns.

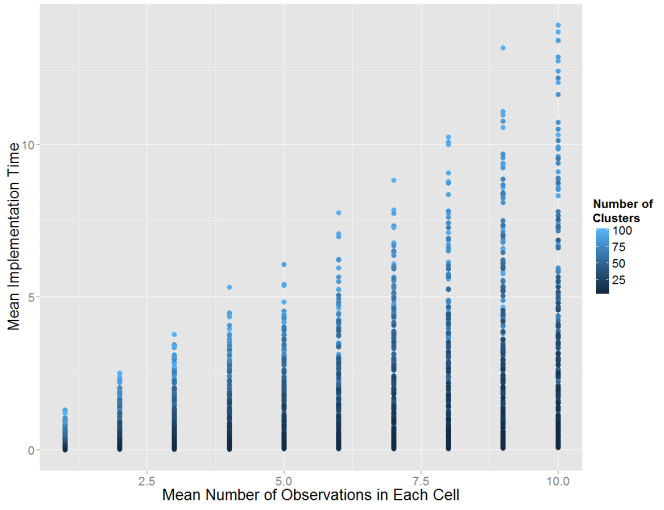
For the multinomial sampling plan, the scatter plot of mean implementation time versus number of observations colored according to number of clusters is very similar to the one given in panel (a) of Figure 1. For the Poisson sampling plan, the scatter plot of mean running time versus mean number of observations in each cell colored according to number of centers is very similar to the one given in panel (b) of Figure 1. For the product multinomial sampling plan, the scatter plot of mean running time versus fixed row totals colored according to number of centers is very similar to the one given in panel (c) of Figure 1. Therefore, these plots are omitted here.

Under the multinomial sampling plan, the mean running time for `rTableICC.2x2xK` is equally affected by number of clusters and number of centers. The number of observations has the primary effect on mean running time. We have long mean running times even for small number of clusters or number of centers if the number of observations is large. Smaller mean running times with high number of centers were recorded for small number of clusters and vice versa. Due to high running times in a small portion of test combinations, the overall distribution of times is right-skewed. The overall median of mean running times is 0.589 seconds with overall median variance of 0.002 and 75% of the mean running times are less than 0.945 seconds over the test combinations. Under the Poisson sampling plan, the mean running time of `rTableICC.2x2xK` increases along with the mean number of observations in each cell. We have high running times for greater number of clusters. The same case is also seen for greater number of centers. The mean number of observations in each cell is the dominant factor on implementation time. The overall distribution of mean running times is right-skewed. The overall median of mean running times is 1.109 seconds with overall median variance of 0.016 and 75% of the mean running times are less than 2.793 seconds over the test combinations. Under the product multinomial sampling plan with fixed row margins, the mean running time for `rTableICC.2x2xK` increases with increasing number of observations in each fixed margin. Also, we have longer running times for both greater number of centers and number of clusters. Rarely, it is also possible to have long running times for a moderate number of clusters or a moderate number of centers. The number of observations in the fixed margins has the primary effect on the mean running time. The overall distribution of mean running times is highly right-skewed due to the outlier value seen in panel (c) of Figure 1. The overall median of mean running times is 0.528 seconds with overall median variance of 0.002 and 75% of the mean running times are less than 1.065 seconds over the test combinations.

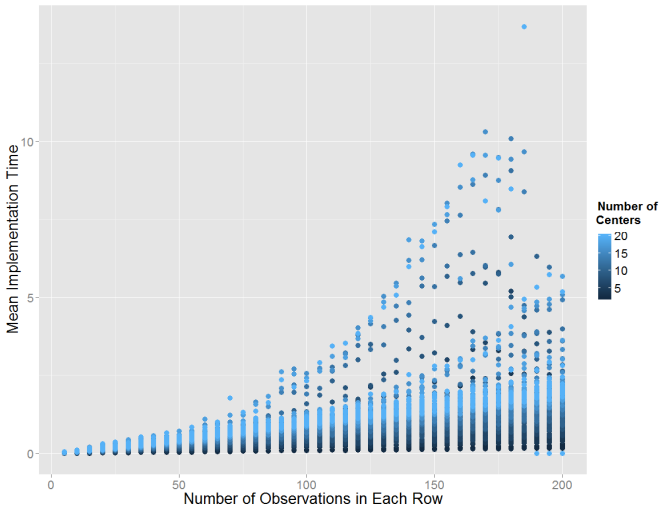
When the function `rTable.2x2xK` was run under the multinomial sampling plan with corresponding test combinations given in Table 1, all of the mean running times were less than 10^{-6} with overall median variance less than 10^{-8}. Therefore, there is no identifiable effect of the test factors on the running time of `rTable.2x2xK`; and hence, no plots are provided for the mean running times of `rTable.2x2xK`. It is possible to record higher running times with a greater number of observations or number of centers. However, setting these parameters to such large values is unreasonable. For the Poisson sampling plan, the maximum mean implementation time over all of the corresponding test combinations in Table 1 is 0.013 seconds. The effect of the number of centers on running time is unobservable. The overall median of mean running times is less than 10^{-6} seconds and the overall average of mean running times is 0.001 seconds with overall median variance less than 10^{-8}. This is due to the nature of the Poisson distribution where in some runs we have a great number of observations in some cells. A similar situation is also seen for the product multinomial sampling plan. Overall the maximum mean running time is 0.013 seconds, the overall average of mean running times is 0.002



(a)



(b)



(c)

Figure 1: Performance of the `rTableICC.2x2xK` function under multinomial, Poisson, and product multinomial sampling plans. Panels (a) and (c) represent mean running time versus number of observations colored according to number of centers for the multinomial and product multinomial sampling plans, respectively. Panel (c) represents mean running time versus mean number of observations in each cell colored according to number of clusters for the Poisson sampling plan.

seconds with overall median variance less than 10^{-8}. The effect of the number of centers is negligible.

For the function `rTableICC.RxC`, plots of mean implementation time versus number of observations and number of clusters colored by number of rows under multinomial, Poisson, and product multinomial samplings are given in Figure 2. Corresponding plots colored by number of columns are very similar to those seen in Figure 2; hence, they are omitted here. For the multinomial sampling plan, mean running times are severely affected by both increasing number of observations and increasing number of rows. However, this is not seen for an increasing number of clusters. We have long mean running times for moderate and small number of clusters. Number of rows (columns) and number of observations are mainly impactful on the running time of `rTableICC.RxC` under the multinomial sampling plan. For the multinomial sampling plan, the overall average of mean running times is 21.312 seconds with median variance of 0.015. The overall median of mean running times is 0.515 seconds, their distribution is highly right-skewed, and 75% of the mean running times are less than 2.999 seconds. For the Poisson sampling plan, the mean running time is mainly affected by the mean number of observations in each cell. Because of the nature of the Poisson distribution, it is possible to obtain long running times even for small number of rows (columns) or clusters. Therefore, we limited the mean number of observations in each cell by 7 in test combinations. The overall average of mean running times is 8.419 seconds with median variance less than 10^{-8}. The overall median of mean running times is 0.047 seconds, their distribution is highly right-skewed, and 75% of the mean running times are less than 0.307 seconds. For the product multinomial sampling plan, the running time is mainly affected by both fixed row counts and number of rows (columns). It is possible to have long running times even for smaller number of clusters if row counts are high. The overall average of mean running times is 0.198 seconds with median variance of $1.33 \cdot 10^{-4}$. The overall median of mean running times is 0.147 seconds, their distribution is right-skewed, and 75% of the mean running times are less than 0.263 seconds.

For the function `rTable.RxC`, we have similar results than for `rTable.2x2xK`. Under multinomial, Poisson, and Product multinomial sampling plans, the overall averages of mean running times are 0.00007, 0.001, and 0.001 with overall median variances less than 10^8, $1.92 \cdot 10^{-5}$, and $1.86 \cdot 10^{-5}$, respectively. The overall medians of mean running times are all less than 10^{-6}. Because we have several outliers in the Poisson and product multinomial sampling plans, the overall average mean running times are greater than 10^{-4}. Due to these numerical results, we cannot identify a significant effect of neither number of rows or columns nor number of observations in cells on the performance of `rTable.RxC`.

In conclusion, the performance of the functions generating tables without ICC is better than those generating tables with ICCs. Running times of both `rTable.2x2xK` and `rTable.RxC` are not notably affected by the values of their arguments and short enough to be used in combination with other Monte Carlo simulation algorithms. Running times of both `rTableICC.2x2xK` and `rTableICC.RxC` are severely affected by the process carried out by the function `compositions` of the package **partitions**. Therefore, their running times are sensitive to inputs and, in general, affected by the total number of individuals to be generated. If generation of a table with a very large total number of individuals is intended, a smaller number of individuals can be generated by a proper scaling on the number of individuals in each cell.

Summary

In this article, we introduced the R package **rTableICC** to generate 2×2×K and R×C contingency tables with and without intraclass-correlated individuals. We described a new approach implemented in functions `rTableICC.2x2xK` and `rTableICC.RxC` for the generation of tables under the presence of intraclass correlations between individuals. Also, we described the function `rDiscrete` for random number generation from empirical probability functions. We provided detailed algorithms working behind the functions and illustrated use and input-output structures of functions in **rTableICC** by numerical examples. Then, we conducted a detailed performance analysis over mean running times of functions `rTableICC.2x2xK`, `rTable.2x2xK`, `rTable.RxC`, and `rTableICC.RxC`. In the performance analysis, we obtained very short running times for the functions `rTable.2x2xK` and `rTable.RxC`, and reasonable running times for the functions `rTableICC.2x2xK` and `rTableICC.RxC`.

As a limitation, when there is ICCs between individuals and the number of rows or columns is greater than 5, functions `rTableICC.2x2xK` and `rTableICC.RxC` may require long running times based on the total number of individuals to be generated. The cause of this situation is the execution time required by the `compositions` function of the package **partitions**. To overcome this limitation, we are planning to decrease complexity of some inner loops of both `rTableICC.2x2xK` and `rTableICC.RxC` functions in forthcoming versions of **rTableICC**.

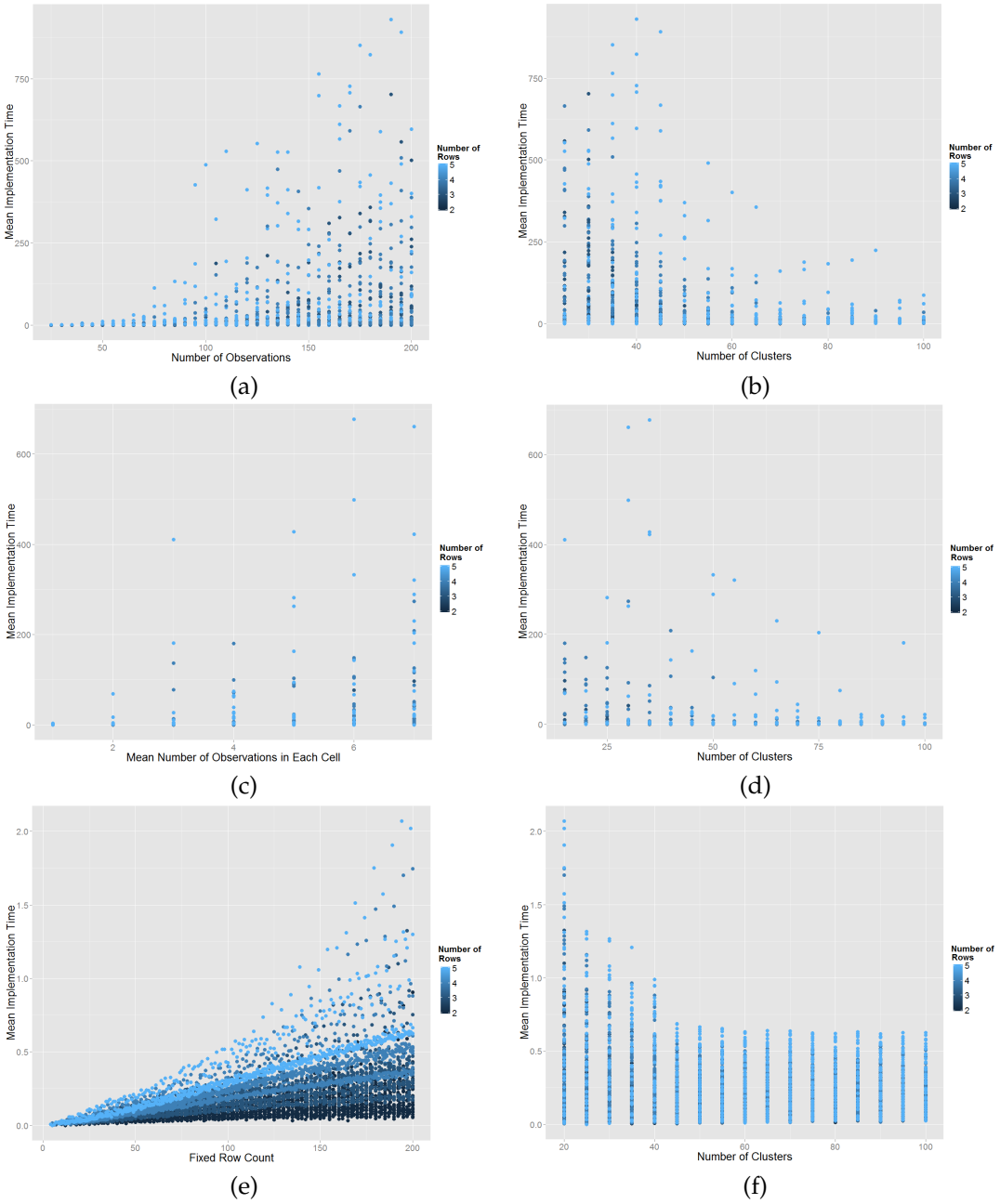


Figure 2: Performance of the `rTableICC.RxC` function under considered sampling plans. Panel (a) shows mean running time versus number of observations colored by number of rows for the multinomial sampling plan. Panel (c) shows mean running time versus mean number of observations in each cell colored by number of rows for the Poisson sampling plan. Panel (e) shows mean running time versus fixed row counts colored by number of rows for the product multinomial sampling plan. Panels (b), (d), and (f) represent mean running time versus number of clusters colored by number of rows for the multinomial, Poisson, and product multinomial sampling plans, respectively.

Bibliography

A. Agresti. *Categorical Data Analysis*. Wiley, New York, 2nd edition, 2002. [p48, 51, 52]

P. Altham. Discrete variable analysis for individuals grouped into families. *Biometrika*, 63:263–269, 1976. [p48, 49]

J. Bi and C. Kuesten. Intraclass correlation coefficient (ICC): A framework for monitoring and assessing performance of trained sensory panels and panelists. *Journal of Sensory Studies*, 27:352–364, 2012. [p48]

Y. Bishop, S. Fienberg, and P. Holland. *Discrete Multivariate Analysis: Theory and Practice*. The MIT Press, Cambridge, 1975. [p48]

C. Borkowf. On multidimensional contingency tables with categories defined by the empirical quantiles of the marginal data. *Journal of Statistical Planning and Inference*, 91:33–51, 2000. [p48]

J. Cohen. The distribution of the chi-squared statistic under clustered sampling from contingency tables. *Journal of the American Statistical Association*, 71:665–670, 1976. [p48]

H. Demirhan. Bayesian estimation of log odds ratios over two way contingency tables with intraclass correlated cells. *Journal of Applied Statistics*, 40:2303–2316, 2013. [p48]

H. Demirhan. *rTableICC: Random Generation of Contingency Tables*, 2015. URL https://CRAN.R-project.org/package=rTableICC. R package version 1.0.3. [p48]

H. Demirhan and C. Hamurkaroglu. Bayesian estimation of log odds ratios from $R \times C$ and $2 \times 2 \times K$ contingency tables. *Statistica Neerlandica*, 62:405–512, 2008. [p48]

R. K. S. Hankin. Additive integer partitions in R. *Journal of Statistical Software*, 16:1–3, May 2006. doi: 10.18637/jss.v016.c01. [p49]

B. Nandram and J. Choi. Bayesian analysis of a two way categorical table incorporating intraclass correlation. *Journal of Statistical Computation and Simulation*, 76:233–249, 2006. [p48]

W. Patefield. Algorithm AS159. An efficient method of generating $R \times C$ tables with given row and column totals. *Applied Statistics*, 30:91–97, 1981. [p48]

Haydar Demirhan
Hacettepe University
Department of Statistics, Beytepe 06800 Ankara
Turkey
haydarde@hacettepe.edu.tr

Maps, Coordinate Reference Systems and Visualising Geographic Data with mapmisc

by Patrick E. Brown

Abstract The **mapmisc** package provides functions for visualising geospatial data, including fetching background map layers, producing colour scales and legends, and adding scale bars and orientation arrows to plots. Background maps are returned in the coordinate reference system of the dataset supplied, and inset maps and direction arrows reflect the map projection being plotted. This is a "light weight" package having an emphasis on simplicity and ease of use.

Introduction

R has extensive facilities for managing, manipulating, and visualising spatial data, with the **sp** (Pebesma and Bivand, 2005) and **raster** (Hijmans, 2015b) packages providing a set of object classes and core functions which numerous other packages have built on. It is fairly straightforward to import spatial data of a variety of types and from a range of sources including: images for map backgrounds; high-resolution pixel grids of surface elevation; and polygons of administrative region boundaries. Large volumes of such data are available for download from sites such as worldgrids.org, gadm.org, and nhgis.org, and map images are freely available from OpenStreetMap.org and other online maps. The first issue often encountered after downloading and importing spatial data is reconciling different coordinate reference systems (CRS's, or map projections). Most repositories of spatial data provide longitude-latitude coordinates, although single-country data sources often use a country-specific map projection (i.e. the UK's Ordinance Survey National Grid) and online maps mostly use the Web Mercator projection. The suitability of a particular map projection will depend on the geographic region being considered and the specific problem at hand.

The **mapmisc** package (Brown, 2016) provides tools for working with projected data which cover the following four areas:

- producing maps with projected data, including scale bars, background images, and inset maps;
- defining and using equal-area map projections for displaying the entire globe;
- creating optimal region-specific map projections where distances are preserved; and
- mapping with colour scales for continuous and categorical data.

This paper will cover each of these points in turn, working through examples and briefly describing the operations by the functions in the **mapmisc** package. An emphasis is given to tidy, intuitive, and reproducible code accessible for students and non-specialists.

Installation and related packages

The two most important packages required for using spatial data in R are the **sp** and **raster** packages, which provide tools and classes for vector data (spatial data on a continuous domain) and raster data (defined on a pixelated grid) respectively. Installing **mapmisc** with `install.packages("mapmisc")` or by using a menu item on a GUI will install **sp** and **raster** if they are not already present. A third important spatial package is **rgdal** (Bivand et al., 2016), which provides methods for re-projecting coordinates and importing spatial data in various file formats. The Geographic Data Abstraction Language (GDAL) underlies **rgdal**, aligning with R's UNIX-like philosophy of combining separate and specialised pieces of software. On most UNIX-based systems, the GDAL and proj4 software must be installed separately prior to installing **rgdal**. All versions of Windows and most versions of MacOS have binary versions of **rgdal** which include the GDAL and proj4 binaries, and **rgdal** can be installed in the same manner as any other R package.

Additional packages used by **mapmisc** are: **RColorBrewer** (Neuwirth, 2014), **classInt** (Bivand, 2015), **rgeos** (Bivand and Rundel, 2016), and **geosphere** (Hijmans, 2015a). These four packages and **rgdal** are not always installed automatically with **mapmisc**, as they are marked as "suggested" packages with **mapmisc** being usable with a reduced level of functionality without them. Three further packages necessary for reproducing the examples in this paper are **dismo** (Hijmans et al., 2016), **maptools** (Bivand and Lewin-Koh, 2016), and **R.utils** (Bengtsson, 2016).

Loading the packages with

```
library("rgdal")
library("mapmisc")
```

also makes **sp** and **raster** available. The remaining packages do not need to be loaded explicitly and will be called by **mapmisc** as needed.

Getting started with spatial data in R

The `getData` function provided by **raster** is able to download a number of useful and interesting spatial datasets. The coastline and borders of Finland can be fetched with

```
finland <- raster::getData("GADM", country = "FIN", level = 0)
```

The object `finland` is a "SpatialPolygonsDataFrame", and Bivand et al. (2013) contains a wealth of information on working with objects of this type. The command

```
plot(finland, axes = TRUE)
```

produces the plot in Figure 1a.

The choice of Finland as an example is due to its being far from the equator, and a useful contrast on the other side of the world is New Zealand. The coastline of New Zealand, obtained with

```
nz <- raster::getData("GADM", country = "NZL", level = 0)
```

includes a number of small outlying islands. The spatial extent of the `nz` object,

```
raster::extent(nz)
```

```
## class       : Extent
## xmin        : -179
## xmax        : 179
## ymin        : -52.6
## ymax        : -29.2
```

spans the entire globe in the x-direction since New Zealand has islands on both sides of the 180° meridian. Finding an appropriate axis limit through trial and error brings one to

```
plot(nz, xlim = c(167, 178), axes = TRUE)
```

and the map in Figure 1b.

The outlying islands can be removed from the `nz` object using the `crop` function in the **raster** package, which in turn calls **rgeos**. Using the `locator` function and a few iterations of trial and error leads to the discovery that a region spanning 160 to 180 degrees longitude and -47 to -30 degrees latitude boxes in the main islands of New Zealand fairly tightly. The parts of New Zealand contained within this box can be extracted by creating an `extent` object and passing it to `crop`.

```
nzClip <- raster::crop(nz, extent(160, 180, -47, -30))
```

The `finland` and `nzClip` objects will be used in the mapping examples which follow.

Working with map projections

This section covers mapping projected data and defining customised map projections. Adding background images, scale bars, and inset maps to plots with the `map.new`, `openmap`, `scaleBar`, and `insetMap` functions is demonstrated in the production of Figure 2. Map projections suitable for displaying the entire globe are constructed with the `moll` function, and along with the `wrapPolys` function Figure 5 is made. Map projections where Euclidean distances from x-y coordinates are useful approximations of shortest distances between points on the globe are obtained with the `omerc` function and used to produce Figure 7.

Spatial data with coordinate reference systems

The spatial coordinates in Figures 1a and 1b are angles of longitudes and latitudes; coordinates which would be equivalent to the two angles of the spherical coordinate system (ρ, θ, ϕ) familiar to mathematicians were it not for the inconvenient fact that the Earth is not spherical. The Earth is rather an oblate spheroid, slightly "squashed" or pumpkin-shaped, and the angles of orientations of lines pointing directly "up" (with reference to the stars) and "down" (as defined by a plumb line

pulled straight by gravity) differ. As a result, various types of long-lat coordinates are in use, with the World Geodesic System (WGS84) used by Global Positioning Systems being the most widespread. The European Petroleum Standards Group (EPSG) catalogue of Coordinate Reference Systems (or CRS's) refers to this system by the code 4326, and this code can be used to create an R object of class "CRS" corresponding to the WGS84 system using the `CRS` function from the **sp** package.

```
sp::CRS("+init=epsg:4326")
```

```
## CRS arguments:
##  +init=epsg:4326 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84
## +towgs84=0,0,0
```

The syntax of `+argument=value` for specifying a CRS comes from the PROJ.4 Cartographic Projections Library, with `+proj=longlat` indicating coordinates are angles and `ellps=WGS84` specifying that the Earth is an ellipsoid with values of the major axis length, minor axis length, and flattening corresponding to the WGS84 specification.

The difficulty angular coordinates pose for interpreting maps or using spatial statistics is that Euclidean distance $\sqrt{x^2 + y^2}$ is not always a useful measure of the distance separating two points. The shortest route between two points on a sphere follows a Great Circle which divides the globe into two equal halves. The distance between two points along this path, the Great Circle distance (see Wikipedia, 2015a), can be computed with a trigonometric formula as implemented in the `spDists` function in **sp**. Euclidean distance will be roughly proportional to Great Circle distance for two points near the equator and reasonably close to one another. In Finland and New Zealand, however, Euclidean distance will over-emphasise the east-west direction since one degree of longitude is a much shorter distance (in kilometres) than one degree of latitude. It is for this reason that Greenland appears larger than India on many maps even though the opposite is true. Most R packages which perform spatial analyses rely on Euclidean distance, including this author's **geostatsp** (Brown, 2015), even though Great Circle distance would be straightforward to implement. Fitting a spatial model with **geostatsp** to data in long-lat coordinates from Finland might uncover directional effects with strong correlation in the east-west direction, which could well be an artefact arising from the over-estimation of east-west distances. The importance of transforming spherical coordinates to a coordinate system where the Euclidean distance is a reasonable approximation to the Great Circle distance should not be under-estimated.

Most countries have an "official" CRS which produces accurate Euclidean distances for specific areas of the globe, one of which is the Finland Uniform Coordinate System having EPSG code 2393. This projection is obtained in R by `CRS` with

```
CRS("+init=epsg:2393")
```

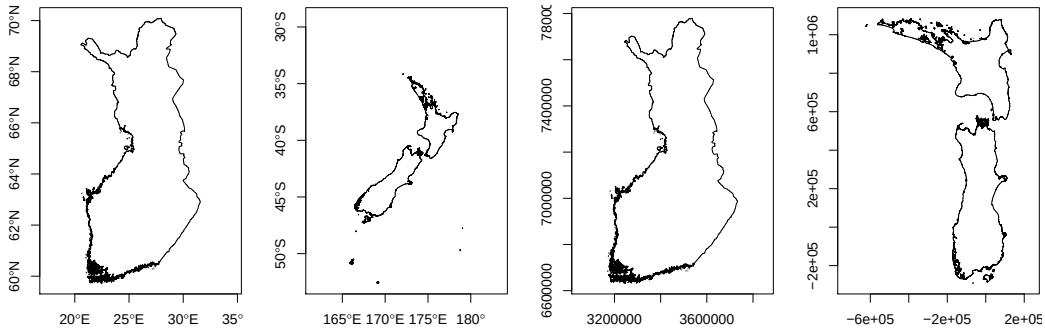
```
## CRS arguments:
##  +init=epsg:2393 +proj=tmerc +lat_0=0 +lon_0=27 +k=1 +x_0=3500000 +y_0=0
## +ellps=intl +towgs84=-96.062,-82.428,-121.753,4.801,0.345,-1.376,1.496
## +units=m +no_defs
```

This is a Transverse Mercator projection (`+proj=tmerc`) with x and y coordinates giving positions on a cylinder containing the earth. The entry `+lon_0=27` indicates that the cylinder touches the Earth along the 27° meridian line. Provided two points are reasonably close to the 27° meridian, Euclidean distance between their Finland Uniform Coordinates will be very close to the true distance between them. The map of Finland can be converted to this coordinate system using `spTransform` from **sp** and **rgdal** with

```
finlandMerc <- spTransform(finland, CRS("+init=epsg:2393"))
```

Figure 1c is the result of plotting this object with `plot(finlandMerc,axes = TRUE)`. Notice the projected map has a wider base and narrower top than the long-lat map in Figure 1a. The coordinates in Figure 1c refer to an origin where the 27° meridian intersects the equator, with the `+x_0=` argument above indicating that 3,500km are added to the x coordinates.

Cylindrical map projections can be constructed from any cylinder containing the Earth, and there is no mathematical requirement to use one of the standard transverse Mercator projections with an EPSG number. For example, a given user might consider the point $(170°\text{E}, 45°\text{S})$ to be an intuitive location for the origin of the transformed map of New Zealand, and thus decide to define a custom CRS with this centre. The cylinder can follow any Great Circle, it need not be a meridian line, and a Great Circle angled 40° clockwise would run the length of the two islands. Cylindrical projections with an angle are termed Oblique Mercator projections (see Snyder, 1987, page 66), and can be constructed with the assistance of **mapmisc**'s `omerc` function. A custom projection with the $(170°\text{E}, 45°\text{S})$ origin and a 40° rotation is obtained by



(a) Finland, Longitude-Latitude

(b) New Zealand, Longitude-Latitude

(c) Finland Transverse Mercator

(d) New Zealand, Oblique Mercator

Figure 1: Basic maps of Finland and New Zealand in different Coordinate Reference Systems.

```
nzCrs <- omerc(c(170, -45), angle = 40)
nzCrs

## CRS arguments:
##  +proj=omerc +lat_0=-45 +lonc=170 +alpha=40 +k=1 +x_0=0 +y_0=0 +gamma=0
## +ellps=WGS84 +units=m
```

The difference between the above and the projection for Finland is the `omerc` in place of `tmerc`, with the additional argument `+alpha=40` specifying an angle. The New Zealand coastline can be projected to this CRS with `spTransform`.

```
nzRot <- spTransform(nzClip, nzCrs)
```

Figure 1d results from executing `plot(nzRot,axes = TRUE)`, and New Zealand has been rotated 40° to a vertical position.

Finding a projection

When choosing a map projection for a dataset, a simple web search of a phrase such as "map projection finland epsg" will often give clear advice as to what the most commonly used national CRS is. A number of tools in **rgdal** can be used to obtain a projection in a more systematic manner. Below the `make_EPSG` function creates a table of all EPSG coded CRS's which **rgdal** supports, and a `grep` command used to show all those projections with "Finland" in its description. The resulting list of projections below confirms that 2393 is a sensible choice.

```
allEpsg <- rgdal::make_EPSG()
allEpsg[grep("Finland", allEpsg$note), 1:2]

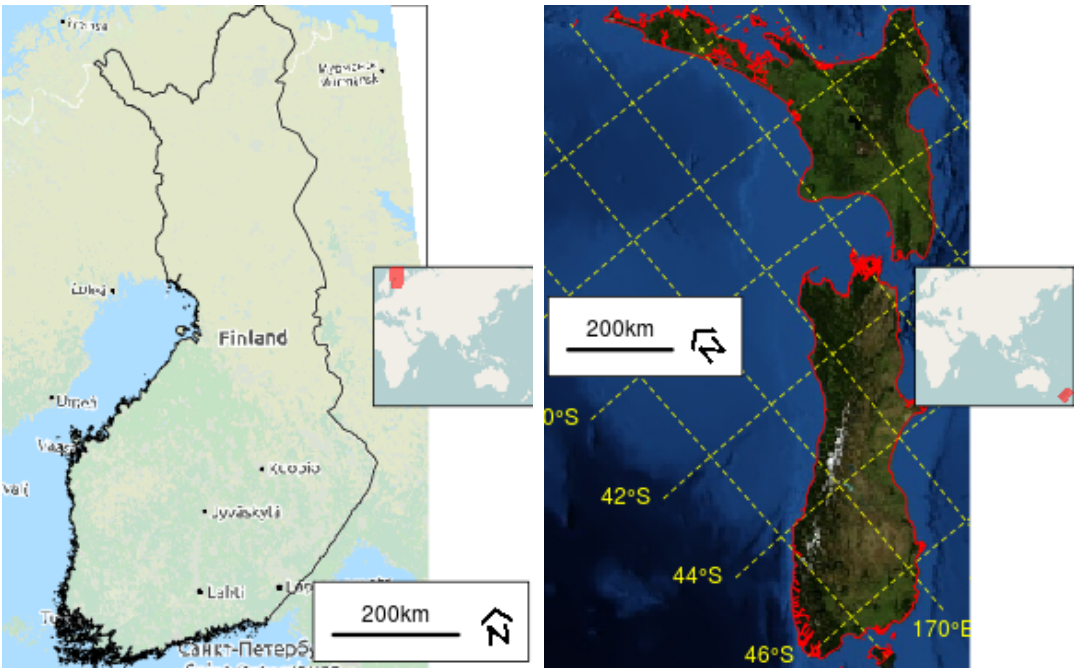
##      code                                                note
## 859  2391                                # KKJ / Finland zone 1
## 860  2392                                # KKJ / Finland zone 2
## 861  2393             # KKJ / Finland Uniform Coordinate System
## 862  2394                                # KKJ / Finland zone 4
## 1853 3386                                # KKJ / Finland zone 0
## 1854 3387                                # KKJ / Finland zone 5
## 4929 3901 # KKJ / Finland Uniform Coordinate System + N60 height
```

A second method for obtaining a CRS is to make a rough guess at a projection string and use `showEPSG` to attempt to find a corresponding EPSG code. Were one to use a Universal Transverse Mercator (or UTM) projection for a map of Finland, a web search for "UTM zone map" shows that Finland lies in UTM zone 35. A proj4 specification of a UTM zone 35 projection will contain `+proj=utm` and `+zone=35`, and `showEPSG` states that the EPSG code 32635 corresponds to an appropriate CRS.

```
rgdal::showEPSG("+proj=utm +zone=35")

## [1] "32635"

CRS("+init=epsg:32635")
```



(a) Finland. Background © OpenStreetMap

(b) New Zealand. Tiles courtesy of MapQuest and © OpenStreetMap

Figure 2: Maps produced using the **mapmisc** package, containing background images, inset maps, and scale bars.

```
## CRS arguments:
##  +init=epsg:32635 +proj=utm +zone=35 +datum=WGS84 +units=m +no_defs
## +ellps=WGS84 +towgs84=0,0,0
```

The definitive resource for information on national map projections is contained in the monthly bulletins of The Imaging and Geospatial Information Society. The `GridsDatums` data set in **rgdal** gives the year and month for each country's entry at www.asprs.org/Grids-Datums.html. The entry for Finland appears in the October 2006 issue.

```
data("GridsDatums", package = "rgdal")
GridsDatums[grep("Finland", GridsDatums$country), ]

##                 country     month year ISO
## 100 Republic of Finland (October) 2006 FIN
```

Mapping projected data

The **mapmisc** functions `openmap`, `map.new`, `scaleBar`, and `insetMap` can be used together to improve on the basic maps in Figure 1, and they are used here to add background images, a scale bar, and an inset map to Figure 2.

Background map images are obtained from the `openmap` function, which downloads image files from OpenStreetMap.org or a number of other sources. The images used in Figure 2 were obtained with

```
nzBg <- openmap(nzRot, path = "mapquest-sat")
finlandBg <- openmap(finlandMerc, path = "landscape")
```

The first argument of `openmap` is used to set both the spatial extent of the map to be retrieved and the CRS the map will be projected to. Any spatial object `x` for which `extent(x)` and `projection(x)` are defined can be provided to `openmap`. The `path =` argument specifies the source of the map, and sample maps from the various sources are shown in Figure 11 in the Appendix. The objects produced by `openmap` are raster objects, converted from image files downloaded from web map servers. The Finland map is an object of class "RasterLayer".

```
finlandBg
```

```
## class       : RasterLayer
## dimensions  : 768, 376, 288768  (nrow, ncol, ncell)
## resolution  : 2747, 2252  (x, y)
## extent      : 2889952, 3922969, 6183827, 7913059  (xmin, xmax, ymin, ymax)
## coord. ref. : +init=epsg:2393 +proj=tmerc +lat_0=0 +lon_0=27 +k=1 +x_0=3500000 +...
## data source : in memory
## names       : landscape
## values      : 1, 1022  (min, max)
```

Notice that the CRS is the same as for the `finlandMerc` object. The New Zealand map is a "RasterStack" with red, green and blue layers.

```
nzBg
```

```
## class       : RasterStack
## dimensions  : 512, 289, 147968, 3  (nrow, ncol, ncell, nlayers)
## resolution  : 5237, 5190  (x, y)
## extent      : -1e+06, 510285, -977333, 1680003  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=omerc +lat_0=-45 +lonc=170 +alpha=40 +k=1 +x_0=0 +y_0=0 +gam...
## names       : mapquest.satRed, mapquest.satGreen, mapquest.satBlue
## min values  :               0,                 0,                0
## max values  :             248,               250,              247
```

The Finland map can be viewed with `plot(finlandBg)`, whereas the three-layered New Zealand map needs the `plotRGB` function for plotting its red, green and blue values as colours.

Figure 2a is produced with the four function calls below.

```
map.new(finlandMerc, 0.8)
plot(finlandBg, add = TRUE)
plot(finlandMerc, add = TRUE)
scaleBar(finlandMerc, "bottomright")
insetMap(finlandMerc, "right", width = 0.3, cropInset = extent(0, 180, -50, 70))
```

The functions run are the following:

- `map.new` initialises a new plot area suitable for showing `finlandMerc`. The second argument set to `0.8` specifies that the left 80% of the plot will contain the map and the right 20% will be reserved for legends or inset maps.
- `plot(finlandBg,add = TRUE)` adds the background map to the existing plot.
- `plot(finlandMerc,add = TRUE)` adds the border of Finland from the `finlandMerc` object.
- `scaleBar` produces the 200km scale and north arrow at the bottom right. The `finlandMerc` object is required to inform `scaleBar` that the Finland Uniform Coordinate System is used, and `scaleBar(CRS("+init=epsg:2393"),"bottomright")` would have achieved the same effect.
- `insetMap` produces the small map to the right, showing in red on the inset map the area covered by the plot. As with `scaleBar` it uses `finlandMerc` to obtain the CRS of the map coordinates. The `width` argument specifies the width of the inset map as a fraction of the plotting region. The `cropInset` argument produces an inset map where New Zealand (at 170°E and 45°S) is in the south-west corner, and the northern limit of Finland (roughly 70°) is encompassed.

The New Zealand map in Figure 2b is produced with similar code.

```
map.new(nzRot, 0.8)
plotRGB(nzBg, add = TRUE)
rgdal::llgridlines(nzRot, col = "yellow")
plot(nzRot, add = TRUE, border = "red")
scaleBar(nzRot, "left")
insetMap(nzRot, "right", width = 0.3, cropInset = extent(0, 180, -50, 70))
```

The use of `plotRGB` in place of `plot` is used for the background map, and the scale bar has been placed at the centre-left. The `llgridlines` function from **rgdal** added latitude and meridian lines in yellow.

The images in Figure 2 have dimensions of 4 by 5 inches, saved as `png` files with 72 pixels per inch. Executing the code above in an interactive R session will likely produce maps with a slightly different appearance unless the graphics window has these same dimensions. This document is produced with **knitr** (Xie, 2015), and the figure dimensions are set with the `fig.height =` and `fig.width =` options to code chunks.

The map images in finlandBg and nzBg were retrieved from OpenStreetMap.org and MapQuest respectively, and although they are free to use and reproduce they must be attributed. The openmapAttribution function produces an attribution for an object produced by openmap or a string valid as a path = argument for openmap. An attribution for nzBg (or "mapquest-sat"), as in the caption for Figure 2, is produced with

```
openmapAttribution(nzBg, short = TRUE)
```

```
##                                   mapquest.sat
## "Tiles courtesy of MapQuest(www.mapquest.com)"
```

The Acknowledgements section of this paper has used this function without the short = TRUE argument.

```
openmapAttribution(finlandBg)
```

```
##   landscape
## "copyright OpenStreetMap.org contributors. Data by OpenStreetMap.org
 #  available under the Open Database License (opendatacommons.org/licenses/odbl),
 #  cartography by Thunderforest.com"
```

The additional argument type = "latex" is used in the source code for this paper, and type = "markdown" is also available.

Projecting background maps

There are a number of potential pitfalls involved when using background map images with projected data, and this section will describe some additional options to openmap and map.new which can help in this regard.

Two undesirable features of Figure 2a are the white triangular section in the top left of the map, and the low resolution and lack of legibility of the names of towns and cities. How this arose can be understood by contrasting the map image retrieved from OpenStreetMap.org in Figure 3a with the Finland Uniform Coordinate System map in Figure 3b. The map in Figure 3a uses coordinates in the Spherical Mercator projection, where a vertically-oriented cylinder is wrapped around a spherical Earth at the equator,[1] and the rectangular area covered by the original map becomes somewhat trapezoidal when projected to the Transverse Mercator coordinates in Figure 3b. The transformation has distorted the text on the image, and the image does not completely cover the black rectangle corresponding to the plotting region of Figure 2a.

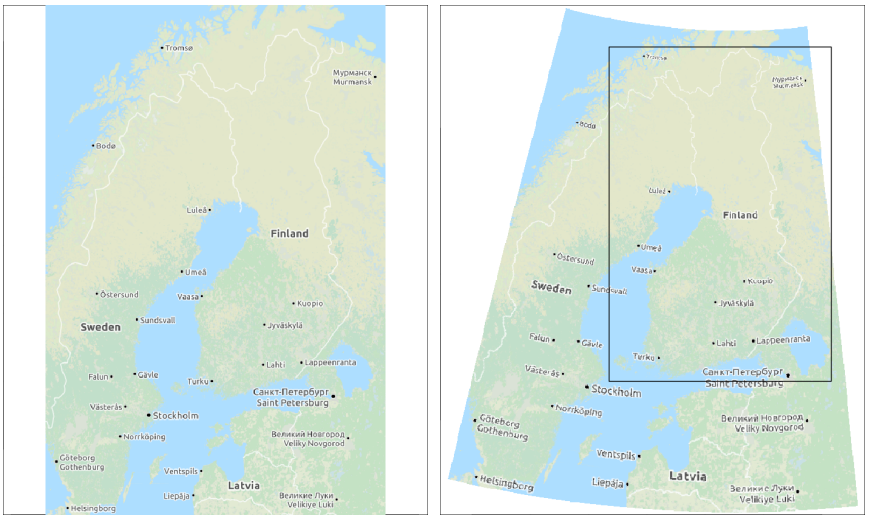
The map images provided by OpenStreetMap.org and elsewhere are available at different zoom levels or resolutions. A map at zoom level 0 is a 256 by 256 pixel image covering the entire world. Zoom level 1 covers the world in 4 "map tiles" of 256 by 256 pixels, and zoom level N consists of 4^N such tiles. The zoom level can be specified directly in openmap with the zoom = argument, or indirectly with the maxTiles = argument. With the default value of maxTiles = 9, openmap will find the highest zoom level where the number of map tiles required to cover the spatial object supplied is at most 9. The finlandBg map has a zoom level of 5 and 6 tiles, giving a 376 by 768 pixel image. This information is contained in an attribute of finlandBg.

```
attributes(finlandBg)$tiles
```

```
## $tiles
## [1] 6
##
## $zoom
## [1] 5
##
## $path
##                                  landscape
## "http://tile.opencyclemap.org/landscape/"
```

```
dim(finlandBg)
```

[1] The Web Mercator, which originated with Google Maps, is in fact slightly different from the Spherical Mercator assigned to map images by **mapmisc**. Maps in a Web Mercator projection are visually identical to a Spherical Mercator, but the Mercator x-y coordinates are hidden and users are shown coordinates which have been converted long-lat (EPSG 4326). The group managing the EPSG codes initially refused to assign a code to the Web Mercator, reportedly saying: "We will not devalue the EPSG dataset by including such inappropriate geodesy and cartography" (Wikipedia, 2016). The Web Mercator was later assigned EPSG code 3857.



(a) Spherical Mercator **(b)** Finland UCS

Figure 3: Map images of Finland in the Spherical Mercator projection and the Finland Uniform Coordinate System (UCS) projection. © OpenStreetMap.

```
## [1] 768 376   1
```

When `openmap` projects the downloaded map tiles to the Finland CRS (using `projectRaster` from the **raster** package), a number of pixels from the original map are lost or put out of position and the text can become mangled.

A partial solution for improving projected images is best illustrated with the rotated CRS used for New Zealand. Figure 4a shows a map of Auckland, New Zealand, in the Spherical Mercator projection provided by OpenStreetMap.org, and the rotated Oblique Mercator projection used earlier is used for the map in Figure 4b. The map images are obtained by creating a "SpatialPoints" object for the location of Auckland in long-lat coordinates and projecting it to the CRS of the `nzRot` object from Figure 2b.

```
aucklandLL <- SpatialPoints(data.frame(x = 174.764204, y = -36.853744),
    proj4string = crsLL)
auckland <- spTransform(aucklandLL, projection(nzRot))
```

`crsLL` is an object in **mapmisc** specifying the WGS84 projection, identical to `CRS("+init=epsg:4326)`. The map in Figure 4b is retrieved below.

```
aucklandBg <- openmap(auckland, buffer = 3000, maxTiles = 4)
```

The `buffer =` argument specifies an additional area around `auckland` which the map should cover (in this case 3km), and specifying `maxTiles = 4` will select the highest zoom level which is able to cover the map region with four or fewer tiles. A map at the same zoom level in the Spherical Mercator projection, for Figure 4a, is obtained next.

```
aucklandBgMerc <- openmap(auckland, zoom = attributes(aucklandBg)$tiles$zoom,
    path = attributes(aucklandBg)$tiles$path, crs = crsMerc)
```

The `crsMerc` object gives the Spherical Mercator projection.

```
crsMerc
```

```
## CRS arguments:
##  +proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0
## +k=1.0 +units=m +no_defs
```

Figure 4c shows only the area within 3km of the `auckland` object, produced by adding the `buffer = 3000` argument to `map.new`.

```
map.new(auckland, buffer = 3000)
plot(aucklandBg, add = TRUE)
scaleBar(auckland, "topleft")
```

One problem with Figure 4b has been resolved, since the displayed area is entirely contained within the map image. The text in Figure 4c is visibly distorted, and the distortion is reduced by increasing

(a) Spherical Mercator **(b)** Oblique Mercator

(c) Oblique, detail **(d)** Oblique, high resolution

Figure 4: Map images of Auckland, NZ, in Spherical Mercator and Oblique Mercator projections. © OpenStreetMap.

the resolution of the image prior to re-projection. The `fact = 4` argument to the call to `openmap` below increases the resolution of the raster by a factor of 4, creating 16 times the number of pixels, yielding the map in Figure 4d.

```
aucklandFine <- openmap(auckland, buffer = 3000,
    zoom = attributes(aucklandBg)$tiles$zoom, fact = 4)
map.new(auckland, buffer = 3000)
plot(aucklandFine, add = TRUE)
scaleBar(auckland, "topleft")
```

Re-projecting rasters is computationally intensive, and `openmap` can require considerable running time when the zoom level or `fact` argument are large.

Equal-area map projections

This section covers producing maps of the entire globe using the functions `moll`, `wrapPoly`, and `gridlinesWrap`. A number of different Coordinate Reference Systems (CRS's) are used to display maps of the world, and Munroe (2011) is a useful introduction to some of the most popular of these. Figure 5a shows the world in a Mollweide projection (Wikipedia, 2015b), an equal-area CRS with the property that the sizes of polygons on the map are roughly proportional to their true surface areas. Figures 5b, 5e and 5f use Mollweide projections with different origins and angles of orientation, projections which are obtainable with **mapmisc**.

Polygons corresponding to the borders of the countries of the world are contained in the `wrld_simpl` object from the **maptools** package, and this object will be used to produce the images in Figure 5. The object is loaded with

```
data("wrld_simpl", package = "maptools")
```

```
wrld_simpl
```

```
## class       : SpatialPolygonsDataFrame
## features    : 246
## extent      : -180, 180, -90, 83.6  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs +towgs84=0,0,0
## variables   : 11
## names       : FIPS, ISO2, ISO3,  UN,           NAME,    AREA,  POP2005, REGION, ...
## min values  :     ,   AD,  ABW,  10, Aaland Islands,       0, 0.00e+00,      0, ...
## max values  :   ZI,   ZW,  ZWE,  96,       Zimbabwe,   99545, 9.94e+04,      9, ...
```

and a selection of R's named colours is assigned to the countries as follows.

```
colVec <- grep("gray|white|snow|ivory|turquoise|blue|[1-3]", colours(distinct = TRUE),
    invert = TRUE, value = TRUE)
wrld_simpl$col <- rep_len(colVec, length(wrld_simpl))
```

The `moll` function creates "CRS" objects for Mollweide projections, which can be used with `spTransform` to compute a Mollweide projection of `wrld_simpl`.

```
mollCrs <- moll()
worldMoll <- spTransform(wrld_simpl, mollCrs)
```

This "CRS" object is specified with a string similar to those seen earlier, with `+proj=moll` being the defining feature.

```
mollCrs
```

```
## CRS arguments:
##  +proj=moll +lon_wrap=0 +lon_0=0 +x_0=0 +y_0=0 +ellps=WGS84 +datum=WGS84
## +units=m +no_defs +towgs84=0,0,0
```

The `moll` function adds two additional attributes to the "CRS" object produced, one of which is the "ellipse" object containing the spatial extent of the Earth in this projection.

```
names(attributes(mollCrs))
```

```
## [1] "projargs" "class"    "crop"     "ellipse"
```

The `map.new` function uses the "ellipse" attribute to define the plotting region and add the light blue background in Figure 5a.

```
map.new(mollCrs, col = "lightblue")
plot(worldMoll, add = TRUE, col = worldMoll$col,
    border = col2html("black", opacity = 0.2))
gridlinesWrap(worldMoll, lty = 2, col = "red")
```

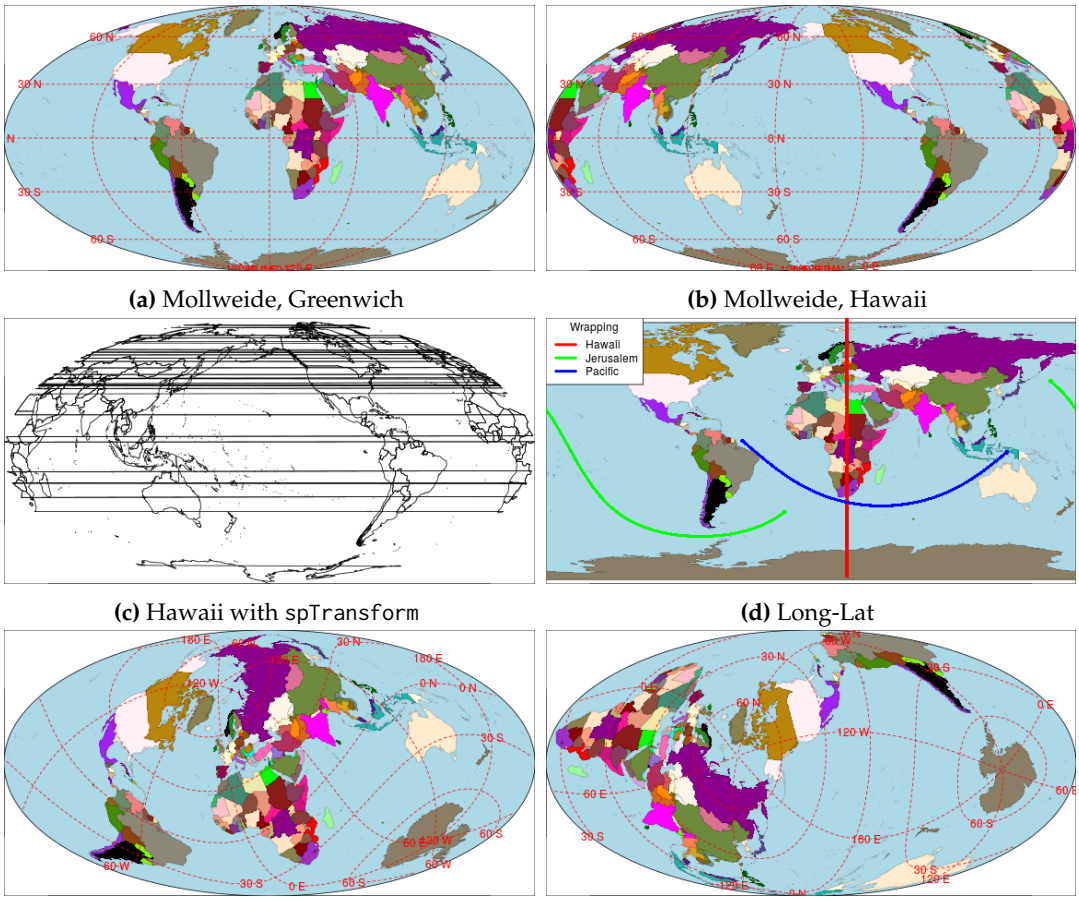
The standard Mollweide projection is symmetric about the 0° Greenwich meridian line and the globe is wrapped or split in the Pacific ocean at the 180° latitude line. Figure 5b is centred around a meridian line passing through Hawaii and splits the Earth along a longitude line passing through Africa and Europe. The CRS for this centred Mollweide projection is produced from the `moll` function, which adds Hawaii's longitude to the `+lon_wrap` and `+lon_0` components of the CRS.

```
(mollHawaii <- moll(geocode("hawaii")))
```

```
## CRS arguments:
##  +proj=moll +lon_wrap=-155.5827818 +lon_0=-155.5827818 +x_0=0 +y_0=0
## +ellps=WGS84 +datum=WGS84 +units=m +no_defs +towgs84=0,0,0
```

A difficulty with this Hawaiian Mollweide projection is that the meridian line along which the world is split runs through many of the polygons in `wrld_simpl`. The `spTransform` function used with this projection produces the polygons in Figure 5c, with horizontal lines connecting the two halves of polygons which have been split. The `wrapPoly` function in **mapmisc** addresses this problem by splitting the affected polygons prior to their projection, using the "crop" attribute of the "CRS" object produced by `moll`.

```
worldHawaii <- wrapPoly(wrld_simpl, mollHawaii)
```



(a) Mollweide, Greenwich **(b)** Mollweide, Hawaii

(c) Hawaii with `spTransform` **(d)** Long-Lat

(e) Oblique Mollweide, Jerusalem **(f)** Oblique Mollweide, Pacific

Figure 5: World maps in Mollweide projections.

The red vertical line in Figure 5d is produced from `attributes(mollHawaii)$crop`, and the function `gDifference` from **rgeos** is called by `wrapPoly` to remove the portion of each polygon intersecting with this line.

A Mollweide projection need not have a north-south orientation, and an Oblique Mollweide projection can be constructed by rotating the globe's long-lat coordinates to produce different origins and orientations. Figure 5e positions Jerusalem at the centre of the Earth (standard practice for cartographers during the middle ages), and at Jerusalem the "up" direction is 35° clockwise of north.

```
(mollOblique <- moll(geocode("jerusalem"), angle = 35))

## CRS arguments:
##  +proj=ob_tran +o_proj=moll +o_lon_p=-42.7134520141079
## +o_lat_p=44.2305998589378 +lon_0=-17.7121046024056
## +lon_wrap=-17.7121046024056 +ellps=WGS84 +datum=WGS84 +units=m +no_defs
## +towgs84=0,0,0
```

This string specifies two projections are to be applied. First, `+proj=ob_tran` rotates the long-lat coordinates to move the north pole to the coordinate (44°N, 42°W). Second, `+o_proj=moll` applies a Mollweide projection to these rotated coordinates using 17.7°W as the central meridian line. These three values (44°N, 42°W, 17.7°W) are obtained by numerical optimisation, attempting to move Jerusalem as close as possible to the origin while preserving the 35° orientation.

Figure 5e is produced by first re-projecting the world map with `wrapPoly`.

```
worldOblique <- wrapPoly(wrld_simpl, mollOblique)
```

Adding red longitude and latitude grid lines with this projection cannot be done with the `llgridlines` function from **rgdal**, as the same horizontal striping seen in Figure 5c will result. The `gridlinesWrap` function from **mapmisc** breaks the grid lines in the same manner as `wrapPoly`, and adds the red graticule lines as follows.

	Truth	Spherical Mercator	Greenwich	Hawaii	Jerusalem	Pacific
Canada	9.98	49.72	9.74	9.79	9.61	9.66
Congo	2.35	2.36	2.34	1.95	2.34	2.34
ratio	4.26	21.08	4.17	5.03	4.11	4.13

Table 1: Surface areas (in millions of km^2) of Canada and Congo, and areas computed for country polygons in the Spherical Mercator projection and four Mollweide projections. Columns refer to the standard Mollweide projection (Greenwich) and the Mollweide centred on Hawaii, Jerusalem, and the north Pacific from Figures 5a, 5b, 5e and 5f.

```
map.new(mollOblique, col = "lightblue")
plot(worldOblique, add = TRUE, col = worldOblique$col,
    border = col2html("black", 0.2))
gridlinesWrap(worldOblique, lty = 2, col = "red")
```

Figure 5f uses an Oblique Mollweide centred in the Pacific ocean with a 85° angle of rotation.

```
crsN <- moll(c(-140, 40), angle = 85)
```

Unlike the red line of the Hawaiian projection in Figure 5d, the green and blue curves showing where the globe is wrapped for the two Oblique Mollweide projections lie in the ocean for the most part.

Table 1 computes the surface areas of Canada and the Congo by using the `gArea` function from **rgeos** on different transformations of the `wrld_simpl` polygons. The first column shows the true surface areas (according to Wikipedia) and the third row shows the ratio of Canada's surface area to the Congo's. The Spherical Mercator projection vastly over-represents the size of Canada and the four Oblique Mollweide projections are consistent in underestimating the area by a small amount. The Congo is poorly served by the Hawaiian projection, which is unsurprising as this projection splits the Congo through the middle.

Custom-optimised Oblique Mercator projections

This section describes the use of the `omerc` function for defining Oblique Mercator projections. Two methods of optimising map projections are implemented, with the angle of rotation chosen to produce the most compact bounding box possible or to preserve distances between a collection of points.

Ad-hoc projections for compact plotting regions

Compact representations minimising the number of void cells (i.e. those falling in the ocean) offer computational advantages when used with statistical models or software requiring data on a grid, such as the Markov random field based methods used in Brown (2015). Figure 6 shows New Zealand rotated in order to produce a compact plotting area, with bounding box having the greatest possible proportion of its area made up of land mass. This projection also minimises the number of vertical inches which Figure 6 takes up on the page, which is the reason a clockwise rotation is used in place of the anti-clockwise rotation from Figure 2b.

The `omerc` function calculates these ad-hoc projections based on the object to be re-projected and a vector of rotation angles. The first argument below is the `nzClip` object containing the cropped boundary of New Zealand, and the origin of the Oblique Mercator projection will be its centroid. A sequence of Oblique Mercator projections using Great Circles angled between 10 and 50 degrees from the north will be formulated, and the bounding box of New Zealand in each projection will be computed. The projection returned by `omerc` uses the angle giving the smallest such box, with the argument `+alpha=` showing an 18.5° angle in this example.

```
(nzRotOptCrs <- omerc(nzClip, seq(10, 50, by = 0.5), post = "wide"))
```

```
## CRS arguments:
##  +proj=omerc +lat_0=-40.565 +lonc=172.502 +alpha=18.5 +k=1 +x_0=0 +y_0=0
## +gamma=90 +ellps=WGS84 +units=m
```

A positive angle rotates the coordinate axes clockwise, which gives the resulting map the appearance of being rotated anti-clockwise. An Oblique Mercator's Great Circle becomes the y-axis of the coordinate system, and a projection should seek to have as much of the area of interest as possible being close to this Great Circle. An optimal projection will therefore have a tall and narrow bounding box, in

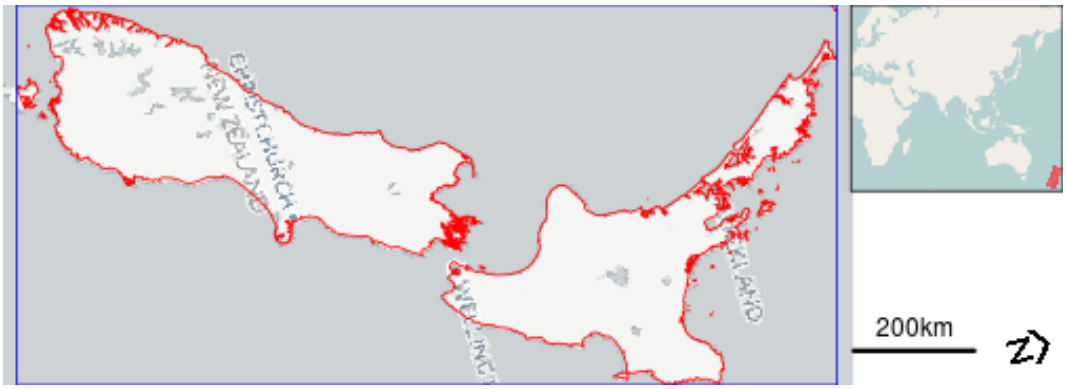


Figure 6: New Zealand in a 18.5° Oblique Mercator projection (—) with bounding box (—). Background © CartoDB, © OpenStreetMap.

this instance the spine of New Zealand should run up and down the y-axis. The limits of 10 and 50 for the sequence of angles above are arbitrary estimates of the amount of anti-clockwise rotation required to sit New Zealand upright. The argument `post = "wide"` in `omerc` specifies that a short and wide bounding box is required, and this is accomplished by rotating the planar x-y coordinates post-projection using the `+gamma=90` element.

New Zealand is re-projected and a new background map in this projection is produced below.

```
nzRotOpt <- spTransform(nzClip, nzRotOptCrs)
nzBgRot <- openmap(nzRotOpt, path = "cartodb", buffer = 20000, fact = 2)
```

Figure 6 is produced with the following.

```
map.new(nzRotOpt, 0.8)
plot(nzBgRot, add = TRUE)
plot(nzRotOpt, add = TRUE, border = "red")
plot(extent(nzRotOpt), add = TRUE, col = "blue")
scaleBar(nzRotOpt, "bottomright", bty = "n")
insetMap(nzRotOpt, "topright", cropInset = extent(0, 180, -50, 70), width = 0.2)
```

Preservation of distances

Here we seek the parameters of an Oblique Mercator projection that would preserve at best the Euclidean distances among a set of points on the surface of the Earth. This is particularly useful for large countries away from the equator and we will use Canada as an example. Figure 7 shows Canada in six different CRS's, the first of which is an Oblique Mercator optimised to preserve distances between 12 provincial and territorial capital cities. Unlike the New Zealand projection, however, the x-y coordinates on the Mercator's cylinder are inverse-rotated to preserve the north-south direction as up-down.

Oblique Mercator projections are defined by an origin and an angle of rotation, only the latter is optimised by the `omerc` function while the origin is set by the user. Here we will set the origin somewhat arbitrarily as the town of Flin Flon, Manitoba, as an alternative to the centroid-based origin used earlier. Residents of Toronto, this author included, affectionately refer to their city as "The Centre of the Universe" [2], but Flin Flon is a more suitable Oblique Mercator centroid for two reasons. First, Flin Flon's latitude is a useful compromise between a northerly centroid able to accommodate the Arctic islands and a centroid close to the populated areas in the south. Second, using Flin Flon as the location where the inverse rotation will preserve the north-south direction as vertical should produce a map with a familiar shape since a portion of the southern border following the 49th parallel will be horizontal.

The locations of the cities required to compute this projection (Flin Flon and the provincial capitals) can be retrieved from Google using the **dismo** package. A wrapper for the `geocode` function in **mapmisc** converts the data retrieved by **dismo** to a "SpatialPointsDataFrame" object.

```
provincialCapitals <- mapmisc::geocode(c("Vancouver, BC", "Edmonton, AB", "Regina",
    "Winnipeg", "Toronto, ON", "Quebec city", "Fredericton", "Charlottetown",
```

[2] Many Canadians residing outside Toronto prefer the somewhat less affectionate nickname of "Hogtown", claiming Torontonians are arrogant and collectively self-centred.

```
    "Halifax, NS", "St Johns, NL", "Iqaluit", "Whitehorse", "Yellowknife"),
    verbose = TRUE)
flinflon <- mapmisc::geocode("Flin Flon")p
```

The names of the provincial capitals and the location of Flin Flon are below.

```
provincialCapitals$name
```

```
##  [1] "Vancouver"     "Edmonton"      "Regina"        "Winnipeg"
##  [5] "Toronto"       "Quebec City"   "Fredericton"   "Charlottetown"
##  [9] "Halifax"       "St John's"     "Iqaluit"       "Whitehorse"
## [13] "Yellowknife"
```

```
coordinates(flinflon)
```

```
##   longitude latitude
## 1      -102     54.8
```

The arguments given in the call to `omerc` below consist of: `x` giving `flinfon` as the origin of the projection; `angle` giving a sequence of rotation angles to be considered; `post = "north"` specifying that the planar coordinates should be inverse-rotated to preserve the north direction; and `preserve` supplying the locations of the provincial capitals other than Iqaluit (the northerly capital of Nunavut) for calculating the distances which the projections will seek to preserve.

```
(cproj <- omerc(x = flinflon, angle = seq(-85, 85, by = 0.25), post = "north",
    preserve = provincialCapitals[provincialCapitals$name != "Iqaluit", ]))
```

```
## CRS arguments:
##  +proj=omerc +lat_0=54.766 +lonc=-101.876 +alpha=-83.5 +k=0.998 +x_0=0
## +y_0=0 +gamma=-83.498 +ellps=WGS84 +units=m
```

The projection above uses an origin (specified by `lat_0` and `longc`) at the coordinates of Flin Flon and a cylinder tangent to the Earth along a Great Circle angled 83.5° anticlockwise (given by `alpha`). The `k = 0.998` component of the CRS scales the coordinates down by a small amount (0.2%). Although pairs of points along the Great Circle will have Euclidean distances and Great Circle distances being equal, the provincial capitals are some distance from this Great Circle and without scaling their Euclidean distances would overestimate true distances by 0.2%. The `gamma = -83.5` component appears as a consequence of having used the `post = "north"` option, rotating the coordinates on the cylinder so that a vertical line passing through the origin contains points which are directly north or south of each other.

Two cities on Figure 7 for whom coordinates have not yet been retrieved are Kitimat, British Columbia, and the hamlet of Grise Fiord, Canada's most northerly civilian settlement[3]. These locations are obtained below.

```
moreCities <- mapmisc::geocode(c("Grise Fiord", "Kitimat"))
cities <- bind(provincialCapitals, moreCities, flinflon)
```

Also shown is the Great Circle which forms the y-axis of the Oblique Mercator. This circle is created using one of the many useful functions in the **geosphere** package.

```
gcircle <- SpatialPoints(geosphere::greatCircleBearing(flinflon@coords, -83.75),
    proj4string = CRS("+init=epsg:4326"))
```

Five projections in addition to the Oblique Mercator are shown in Figure 7: a Lambert Conformal Conic projection used by the Atlas of Canada (EPSG code 3347); a Two-Point Equidistant projection based on the cities of Edmonton and Toronto (obtained from `tpeqd`); a Mollweide projection centred on Flin Flon; the Spherical Mercator used by OpenStreetMap.org and other internet sites; and longitude-latitude or angular coordinates. A list is created containing these four projections using **mapmisc**'s `tpeqd` function in part.

```
crsList <- list("Oblique Merc" = cproj, "Lambert" = CRS("+init=epsg:3347"),
    "2pt Equidist" = tpeqd(cities[cities$name %in% c("Edmonton", "Toronto"), ]),
    "Mollweide" = moll(flinflon, angle = 0), "Spherical Merc" = mapmisc::crsMerc,
    "Long-Lat" = mapmisc::crsLL)
```

The cities and Great Circle are transformed to each of the projections with `mapply`.

[3]Readers should be aware that Grise Fiord resulted from a government-run resettlement program in the 1950's, and involved the deception and neglect of the indigenous people who were relocated to the hamlet.

(a) Oblique Merc **(b)** Lambert

(c) 2pt Equidist **(d)** Mollweide

(e) Spherical Merc **(f)** Long-Lat

Figure 7: Canada and selected cities in five map projections, along with the Great Circle defining the Oblique Mercator projection (· · ·). Background from Natural Resources Canada.

```
citiesT <- mapply(spTransform, CRSobj = crsList, MoreArgs = list(x = cities))
```

Code for retrieving background maps and for producing Figure 7 appears in the Appendix.

Notable features in Figure 7 are the Spherical Mercator's overemphasis of the high Arctic regions, the downward slope of the Lambert projection, the upward slope of the Two-Point Equidistant map, and the "skinny" appearance of the Mollweide. The Oblique Mercator's Great Circle, passing through Flin Flon, Halifax and Kitimat at a nearly horizontal $-83.5°$, is necessarily a straight line in the Oblique Mercator projection and is reasonably straight in three other projections. The yellow longitude-latitude lines and three north arrows on each plot show that only the Spherical Mercator

	Oblique Merc	Lambert	2pt Equidist	Mollweide	Spherical Merc
south					
mean	−0.05	−0.23	−0.13	−6.68	54.19
sd	0.13	1.03	0.34	5.22	8.01
north					
mean	1.74	−2.54	0.54	−1.10	131.43
sd	1.65	0.32	0.81	6.58	43.82

Table 2: Mean and standard deviation of the percentage by which Euclidean distance overestimates Great Circle distance for various map projections. The bottom two rows involve distances involving at least one of Grise Fiord, Iqaluit, Whitehorse and Yellowknife, with the top two rows including only distances not involving any of these four northern locations.

and long-lat projections have the property that north is in the vertical direction throughout the map.

Table 2 compares Euclidean distance to true (or Great Circle) distance for each of the map projections, with the percentage by which the former overestimates the latter for each pair of cities summarised. The first two rows give the mean and standard deviation for the percentage of overestimation of distances between the southern locations (which exclude Grise Fiord, Iqaluit, Whitehorse, and Yellowknife). The Oblique Mercator's underestimation by 0.05% betters the other projections, and the Mollweide's 6.7% underestimation is excusable given the projection's aim of preserving areas rather than distances. The Lambert's 0.23% underestimation appears respectable though the comparatively large standard deviation of 1.03% indicates several city pairs with Euclidean distances deviating several percent from their true separations. Distances involving at least one of the four northern locations are less accurate for all projections (the Mollweide notwithstanding), and the Oblique Mercator's 1.7% overestimation betrays the fact that these northern points were not part of `omerc`'s optimisation criteria. The Two-Point Equidistant projection is notable in its consistency and accuracy, and the Lambert projection does not appear to be a CRS which statisticians should consider using for points in Canada when accuracy of Euclidean distances is the primary concern.

Maps with colour scales

A separate set of facilities in **mapmisc**, complementing but disjoint from the tools related to map projections, assists with the use of colour scales and legends for maps in R. The **RColorBrewer** package gives R users access to the popular ColorBrewer collection of palettes, all of which are displayed in Table 4 in the Appendix. These colours can be used with the functions in the **classInt** package to create colour scales, and the venerable `legend` function is extremely versatile in its ability to create map legends. The steps involved in defining and using colour scales have been streamlined and consolidated into the `colourScale`, `legendBreaks` and `legendTable` functions in **mapmisc**. The process of creating suitable bins, assigning colours to data points based on these bins, specifying transparency when overlaying colours on background maps, and displaying the intervals and colours in a legend has been simplified as much as possible in **mapmisc**, although at least five separate lines of R code are required to produce a single map.

Colours with polygon data

Consider, as a motivating example, the task of visualising the spatial variation in fertility rates in Europe using data available from the statistical office of the European Union (EUROSTAT). EUROSTAT divides the territory of the European Union and several adjoining nations into statistical units organised in a hierarchical system named "nomenclature des unités territoriales statistiques" and given the memorable acronym of NUTS. Social and demographic data on the statistical units are available at `http://ec.europa.eu/eurostat/data/database`, and their boundaries can be obtained from `http://ec.europa.eu/eurostat/web/gisco/geodata/reference-data/administrative-units-statistical-units`. Figure 8 shows the fertility rate in 2010 for 456 units of level 2 in NUTS.

Code in the Appendix for downloading and merging the boundary files and fertility rates produces a "SpatialPolygonsDataFrame" object `euroF`. Each polygon in the object is a territorial unit, and fertility rates for each unit are provided for the years 2001 to 2012 inclusive.

```
euroF
```

```
## class       : SpatialPolygonsDataFrame
```

```
## features    : 456
## extent      : -24.5, 44.8, 34.6, 71.2  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +ellps=GRS80 +no_defs
## variables   : 29
## names       : NUTS_ID, STAT_LEVL_, SHAPE_Leng, SHAPE_Area, age.geo.time, X2013, ...
## min values  :      AT,          0,      0.134,   1.00e+00,     TOTAL,AT, 0.96 , ...
## max values  :    UKN0,          2,      9.946,   9.96e-01,   TOTAL,UKN0, 3.75 , ...
```

The polygons are transformed to the European Terrestrial Reference System (EPSG code 3034) below.

```
euroF <- spTransform(euroF, CRS("+init=epsg:3034"))
```

Colours in Figure 8 relate to the `X2010` variable in `euroF` (fertility in 2010), with the colours themselves taken from the "Spectral" palette of **RColorBrewer**. Below the `colourScale` function creates eight bins (`breaks = 8`) for values of `euroF$X2010`, reversing the colours so that blues correspond to low values and reds are large values (`rev = TRUE`).

```
ecol <- colourScale(euroF$X2010, col = "Spectral", breaks = 8, rev = TRUE,
    style = "jenks", dec = 1, opacity = 0.5)
```

The `style` argument controls how the breaks are computed, and the `"jenks"` option corresponds to the "natural breaks" algorithm from Jenks and Caspall (1971) (see also Pebesma and Bivand, 2005, Section 3.5.2) implemented in the `classIntervals` function of the **classInt** package. A number of clustering algorithms for defining break points are provided by `classIntervals`, and the options for the `style =` argument described in the help files for `classIntervals` are all available using the identically named argument in `colourScale`. The break points are rounded to one decimal place as a consequence of the `dec = 1` argument, and the `opacity = 0.5` option gives the plotted colours 50% transparency.

The result of a call to `colourScale` is a list containing the break points for bins (`breaks`), colours associated with the bins (`col`), and a vector of length 456 (`plot`) with one colour per region.

```
names(ecol)
```

```
## [1] "col"        "breaks"     "colOpacity" "plot"
```

```
ecol$breaks
```

```
## [1] 1.0 1.3 1.4 1.6 1.8 2.0 2.4 2.9 3.8
```

```
ecol$col
```

```
## [1] "#3288BD" "#66C2A5" "#ABDDA4" "#E6F598" "#FEE08B" "#FDAE61" "#F46D43"
## [8] "#D53E4F"
```

```
length(ecol$plot)
```

```
## [1] 456
```

The `breaks` and `col` elements are used for displaying a legend, whereas the `plot` element can be passed as a `col =` argument when running `plot` on a "SpatialPoints" or "SpatialPolygons" object. Following the retrieval of the background map with

```
euroMap <- openmap(euroF, path = "osm-no-labels")
```

Figure 8, minus the country names, can be produced as follows.

```
map.new(euroF)
plot(euroMap, add = TRUE)
plot(euroF, col = ecol$plot, add = TRUE, border = "lightgrey")
legendBreaks("topright", ecol, title = "fertility", bg = "white")
scaleBar(euroF, "bottom", bty = "n")
```

Notice the `col = ecol$plot` argument specifying the colours with which each region is filled, and that the borders of each region were given by `border = "lightgrey"`. The legend on the right is added with `legendBreaks`, which passes most of its arguments to the standard `legend` function from the **graphics** package. The role of `legendBreaks` is to ensure the 9 numeric break points are printed near the boundaries between the coloured squares (as opposed to aligned with their centres).

The country names in Figure 8 are taken from the `wrld_simpl` object used earlier, although all countries smaller than Albania have been removed in order to keep the map from becoming too crowded. Below a portion of the globe in the northern hemisphere is cropped from `wrld_simpl` and subsequently re-projected to the European CRS.

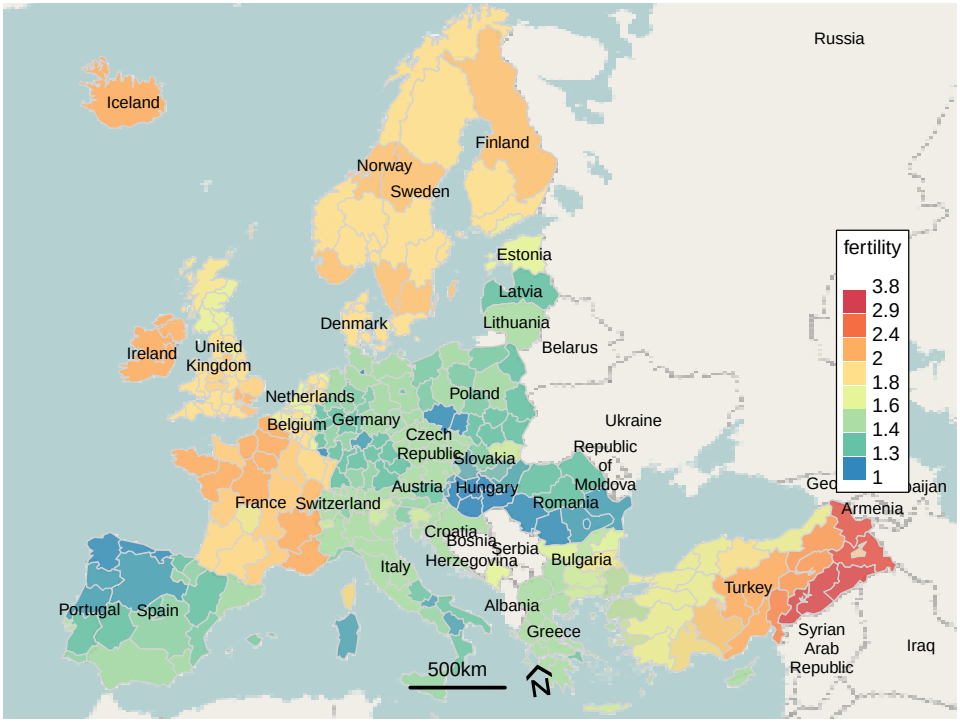


Figure 8: Fertility in Europe by NUTS. Background © OpenStreetMap, © EuroGeographics for the administrative boundaries.

```
data("wrld_simpl", package = "maptools")
worldCrop <- raster::crop(wrld_simpl, extent(-20, 100, 0, 90))
worldE <- spTransform(worldCrop, projection(euroF))
```

The areas of the countries are computed from the map `worldMoll` in the Mollweide projection, merged into the `worldE` object, and all countries at least as large as Albania are retained.

```
worldE$area <- rgeos::gArea(worldMoll, byid = TRUE)[as.character(worldE$ISO3)]
worldE <- worldE[worldE$area >= worldE@data[worldE$NAME == "Albania", "area"], ]
```

The names are added to the plot with the `text` function.

```
text(worldE, labels = worldE$NAME, cex = 0.8)
```

Using the `style = "jenks"` argument for `colourScale` triggers a clustering algorithm in function `classIntervals` which can be time consuming for large datasets. The `"quantile"` and `"equal"` options for the `style` argument use quantiles and equally spaced break points respectively, with the latter able to have breaks equally spaced on a transformed scale (i.e. log or square root). Some examples appear below.

```
colourScale(euroF$X2010, breaks = 8, style = "quantile", dec = 1)$breaks

## [1] 1.0 1.4 1.5 1.7 1.9 2.0 3.8

colourScale(euroF$X2010, breaks = 8, style = "equal")$breaks

## [1] 1.04 1.43 1.82 2.21 2.60 2.99 3.38 3.77

colourScale(euroF$X2010, breaks = 8, style = "equal", transform = "log")$breaks

## [1] 1.04 1.25 1.50 1.81 2.17 2.61 3.14 3.77

colourScale(euroF$X2010, breaks = 8, style = "equal", dec = 0)$breaks

## [1] 1 2 3 4
```

The final set of break points has a `dec = 0` argument, and although 8 breaks have been requested only 4 unique breaks remain after rounding to the nearest integer.

Rasters and colour scales

Figure 9 shows two elevation maps of New Zealand produced with the assistance of the `colourScale` and `legendBreaks` functions. The **raster** package provides versions of the `plot` function for use with rasters, and using `colourScale` with rasters is slightly different from its use in the previous section.

The elevation data is obtained using the **raster** package's `getData` function.

```
nzAltFull <- raster::getData("alt", country = "NZL", keepzip = TRUE)
```

The `nzAlt` object is a list of two elements, with elevation in two disjoint areas which comprise New Zealand. The first element includes the main island, its CRS is incomplete (at the time of writing) and should be re-specified.

```
nzAlt <- nzAltFull[[1]]
projection(nzAlt) <- CRS("+init=epsg:4326")
```

This raster includes a number of outlying islands, and its size becomes more manageable after the main island is extracted using the previously computed `nzClip` object.

```
dim(nzAlt)
```

```
## [1] 2244 1572    1
```

```
nzAlt <- raster::crop(nzAlt, extent(nzClip))
dim(nzAlt)
```

```
## [1] 1544 1458    1
```

This smaller raster is now re-projected to the rotated Oblique Mercator projection used earlier, with the `filename =` argument allowing for the resulting data to be stored as a file in R's working directory rather than in memory.

```
nzAltRot <- projectRaster(nzAlt, crs = projection(nzRot), filename = "nzAltRot.grd")
dim(nzAltRot)
```

```
## [1] 2424 3069    1
```

Re-projecting the raster has made it larger, as the rectangular bounding box of `nzAlt` becomes a diamond when rotated and `nzAltRot` has a bounding box large enough to contain this diamond. The `crop` function is used to pare this raster down to the same extent as the `nzRot` object.

```
(nzAltCrop <- raster::crop(nzAltRot, extent(nzRot)))
```

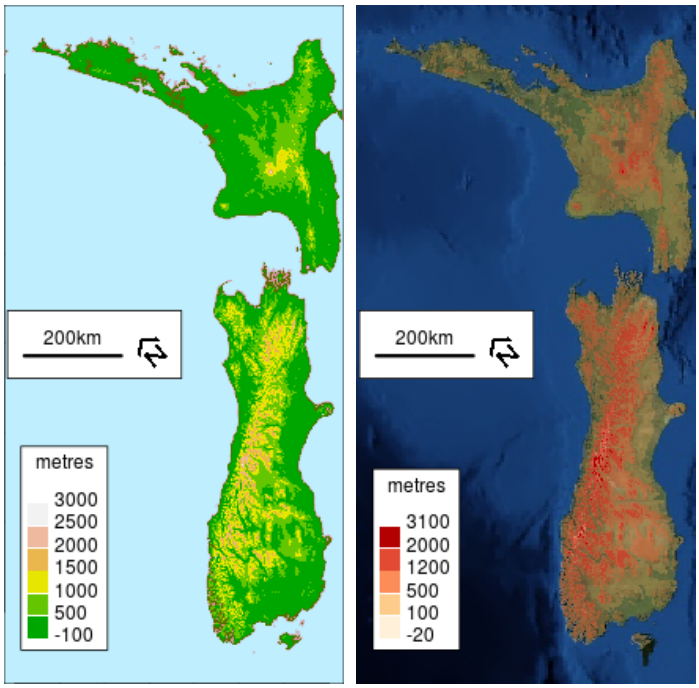
```
## class       : RasterLayer
## dimensions  : 1895, 1369, 2594255  (nrow, ncol, ncell)
## resolution  : 554, 727  (x, y)
## extent      : -624719, 133707, -291892, 1085773  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=omerc +lat_0=-45 +lonc=170 +alpha=40 +k=1 +x_0=0 +y_0=0 +gam...
## data source : in memory
## names       : NZL1_msk_alt
## values      : -11.1, 2960  (min, max)
```

Raster images are typically large, with `nzAltCrop` having over 2 million cells, and computing break points using all of the data points would be time consuming. The **raster** package provide easy access to the maximum and minimum values (-11 and 2960 above), which makes equally spaced break points (`style = "equal"` below) quick to compute. All other styles of breaks are computed using a sample of 20,000 cells taken using **raster**'s `sampleRegular` function.

```
nzAltCol <- colourScale(nzAltCrop, breaks = 7, col = terrain.colors, style = "equal",
    dec = -2)
```

Here the `col =` argument was given the `terrain.colors` function, and any function accepting a single integer argument and returning a vector with the specified number of colours would suffice. Below a second colour scale is computed using the `"OrRd"` colour palette and a supplied vector of breaks.

```
nzAltTrans <- colourScale(nzAltCrop, breaks = c(-20, 100, 500, 1200, 2000, 3100),
    col = "OrRd", style = "fixed", opacity = c(0.2, 1))
```



(a) `terrain.colors()`

(b) `"OrRd"` colours with opacity. Tiles courtesy of MapQuest.

Figure 9: Elevation maps of New Zealand using colours from `terrain.colors` and the `"OrRd"` colours from **RColorBrewer**.

Here the `opacity` argument has been given a vector of length 2, giving the opacity of the first colour and last colour respectively with intervening colours having opacities which are linear interpolations of these values.

Figure 9a is produced with code below.

```
par(bg = "lightblue1")
map.new(nzRot)
par(bg = "white")
plot(nzAltCrop, col = nzAltCol$col, breaks = nzAltCol$breaks, legend = FALSE,
    add = TRUE)
plot(nzRot, add = TRUE, border = col2html("red", 0.4))
legendBreaks("bottomleft", nzAltCol, title = "metres", bg = "white")
scaleBar(nzRot, "left")
```

No background map is present, though the background has been set to a sea-like colour. The line of code beginning with `plot(nzAltRot,...` adds the elevation data to the map. The `col` and `breaks` elements of the colour scale `nzAltCol` are passed as identically named arguments of the `plot` method from the **raster** package. The `legend = FALSE` argument prevents `plot` from adding its own legend, which would not fit well with this map as it always appears on the right.

The code for Figure 9b is below.

```
map.new(nzRot)
plotRGB(nzBg, add = TRUE)
plot(nzAltCrop, col = nzAltTrans$colOpacity, breaks = nzAltTrans$breaks,
    legend = FALSE, add = TRUE)
legendBreaks("bottomleft", nzAltTrans, title = "metres")
scaleBar(nzRot, "left")
```

Here the `colOpacity` element of the colour scale, which has 2-digit opacity levels appended to each specified colour, is passed as `col =` argument to allow the satellite photo beneath to be viewed.

Colours with categorical data

Figure 10 maps land categories in Africa, and `colourScale` has been used to assign colours to 10 of the 20 land categories present. The way the **raster** package treats categorical data (a `factor` in R parlance)

requires a slightly different use of `colourScale` from the previous section, and legends suitable for categorical data can be produced by `legendBreaks` and the related function `legendTable`.

The land cover data originate from the European Space Agency, and are redistributed by worldgrids.org. The file is provided in `.tif` format, compressed as a (`.gz`) file. The code below downloads, unzips, and loads the data into R.

```
landUrl <- "http://www.worldgrids.org/lib/exe/fetch.php?media=glcesa1a.tif.gz"
gzfile <- "glcesa1a.tif.gz"
tiffile <- "glcesa1a.tif"
download.file(landUrl, gzfile)
R.utils::gunzip(gzfile, overwrite = file.exists(tiffile), remove = FALSE)
land <- raster(tiffile)
```

This `land` raster object spans the entire globe, and a portion of central Africa containing both Liberia and Tanzania is extracted with the help of the `wrld_simpl` object used earlier.

```
worldSub <- wrld_simpl[grep("Liberia|Tanzania", wrld_simpl$NAME), ]
worldSub <- spTransform(worldSub, projection(land))
landSub <- raster::crop(land, extend(extent(worldSub), 5))
```

The `extend` function has added an additional 5 units (in this case degrees latitude and longitude) to the region to be extracted.

A text file containing a list of land categories and their numeric identifiers is posted at worldgrids.org, it is retrieved and loaded below.

```
download.file("http://www.worldgrids.org/lib/exe/fetch.php?media=glcesa.txt",
    "landLevels.txt")
landTable <- read.table("landLevels.txt", header = TRUE, sep = "\t",
    stringsAsFactors = FALSE)
```

This table can be used with `colourScale` to produce map colours and a legend, although the table must first be modified as `colourScale` expects it to include a numeric `ID` column and a column called `label` of descriptions. The land categories are assigned integer values in the raster

```
unique(landSub)
```

```
##  [1]  11  14  20  30  40  50  60  70  90 100 110 120 130 140 150 160 170 180 190
## [20] 200 210
```

which correspond to the trailing digits of the `DESCRIPTION` column of the table.

```
landTable[1:2, ]
```

```
##      COLOR                           NAME DESCRIPTION MINIMUM MAXIMUM
## 1 15790250 No data (burnt areas, clouds,)        CL11    10.1    11.1
## 2  6619135              Rainfed croplands        CL14    11.1    14.1
```

The codes are converted into the numeric `ID` variable and the `NAME` column tidied up with a complex gsub statement and saved as `label` below.

```
landTable$ID <- as.numeric(gsub("^CL", "", landTable$DESCRIPTION))
landTable$label <-
    gsub("Closed to open| - [[:print:]]+|\\(([[:digit:]]|[[:punct:]]|m)+\\)", "",
    landTable$NAME)
landTable$label <- trimws(landTable$label)
landTable[c(1:3, 20:23), c("ID", "label")]
```

```
##     ID                                                    label
## 1   11                          No data (burnt areas, clouds,)
## 2   14                                       Rainfed croplands
## 3   20 Mosaic cropland  / vegetation (grassland/shrubland/forest)
## 20 200                                              Bare areas
## 21 210                                             Water bodies
## 22 220                                   Permanent snow and ice
## 23 230                          No data (burnt areas, clouds,)
```

The `labels` argument is used to provide information on categories and labels to `colourScale`, and it requires a `data.frame` with columns named `ID` and `label` giving category identifiers and descriptions respectively.

```
landLevels <- colourScale(landSub, breaks = 10, style = "unique", col = "Set3",
    exclude = c(11, 210, 220, 230), labels = landTable)
```

Here 10 colours have been requested from the "Set3" palette, with style = "unique" specifying that the data are categorical rather than continuous. The 10 most common land types will be assigned colours, although the exclude = argument specifies that several categories (i.e. 11 no data, 210 water bodies, 200 bare areas) will not be colour-coded regardless of how prevalent they are.

```
names(landLevels)
```

```
## [1] "col"         "breaks"      "colOpacity"  "colourtable" "colortable"
## [6] "levels"      "legend"
```

The colourtable and levels elements above were not present when colourScale was used in the previous section, and these elements will be produced whenever a labels = argument has been provided. The levels element is a data frame with one row per land category and columns with labels and colours, and is compatible with the **raster** package's facilities for categorical variables. A categorical raster has a list of data frames, one for each raster layer, accessible by executing levels(landSub). The land categories are the first and only layer of the landSub raster and the table produced by colourScale is added to the first element of the levels list below.

```
levels(landSub)[[1]] <- landLevels$levels
```

The colourtable object in landLevels is a vector of colours associated with each numeric category, with NA's for those categories for which no colour has been assigned (ID 201, Water Bodies for example). It will be used by **raster**'s plotting functions if it has been added to the legend@colortable slot of a raster as follows.

```
landSub@legend@colortable <- landLevels$colourtable
```

The American spelling "color" is used by the majority of R packages, despite the Guidelines for Rd files stating: "For consistency, aim to use British (rather than American) spelling." This author, being Canadian, requires "color" and "colour" to be interchangeable and provides landLevels$colortable (and a colorScale function) to this effect.

Figure 10 includes the "stamen-toner" web map as a foreground (rather than background) layer with country borders and names. The tonerToTrans function converts the white pixels in the "stamen-toner" map to transparent (and greys to semi-transparent), allowing the map to be added after and on top of the land category image.

```
landMap <- openmap(landSub, path = "stamen-toner")
landMapTrans <- tonerToTrans(landMap)
```

Figure 10 is produced as follows.

```
map.new(landSub)
plot(landSub, add = TRUE)
plot(landMapTrans, add = TRUE)
legendBreaks("bottomleft", landSub, ncol = 2, width = 25, lines = 3,
    text.col = "yellow", cex = 0.8, pt.cex = 3, inset = 0, bty = "n")
```

The landSub raster, having had landLevels$colourtable and landLevels$levels attached to it, is the only object required by the plot and legendBreaks functions. The legendBreaks function passes the ncol, cex and inset arguments to the legend function, arguments specifying two columns, slightly smaller than normal text, and a position flush to the bottom left respectively. The width = 25 argument inserts line breaks in the legend labels after 25 characters, and the lines = 3 argument causes only the first three lines to be displayed.

An alternative to including the legend on the plot is to create an "in-line" legend as in the caption to Figure 10. The legendTable function assists in this task, with the code

```
legendTable(landSub, collapse = "; ", type = "latex")
```

used in this instance. Table 3 is created with the following.

```
Hmisc::latex(legendTable(landSub, type = "latex"), file = "", rowname = NULL,
    where = "htb", caption.loc = "bottom", colheads = FALSE,
    caption = "Land categories in Figure \\ref{fig:plotLand}")
```

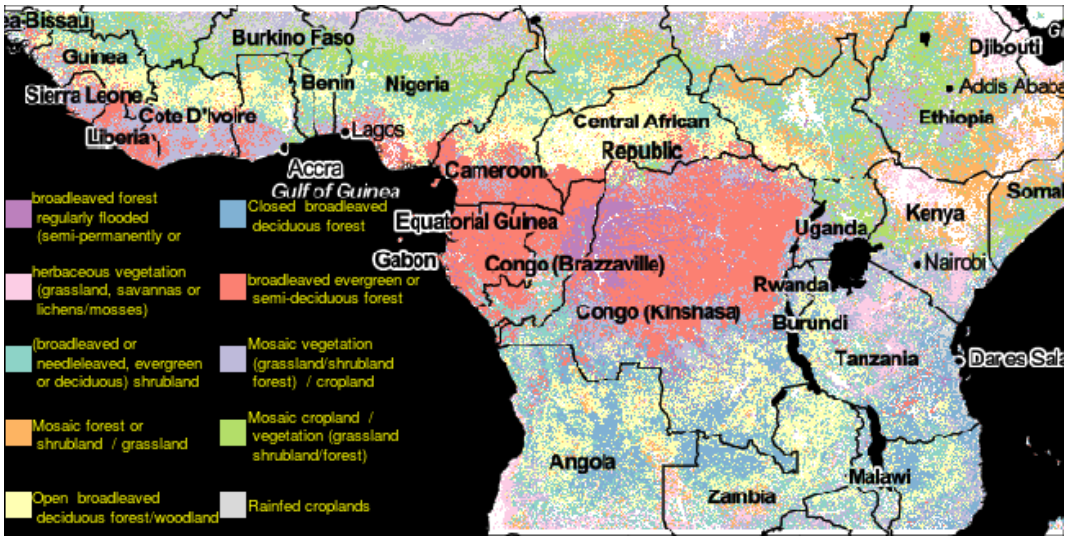


Figure 10: Land categories in central Africa: Rainfed croplands ( ); Mosaic cropland / vegetation (grassland/shrubland/forest) ( ); Mosaic vegetation (grassland/shrubland/forest) / cropland ( ); Broadleaved evergreen or semi-deciduous forest ( ); Closed broadleaved deciduous forest ( ); Open broadleaved deciduous forest/woodland ( ); Mosaic forest or shrubland / grassland ( ); (Broadleaved or needleleaved, evergreen or deciduous) shrubland ( ); Herbaceous vegetation (grassland, savannas or lichens/mosses) ( ); Broadleaved forest regularly flooded (semi-permanently or temporarily) ( ). Background © Stamen Design.

Rainfed croplands
Mosaic cropland / vegetation (grassland/shrubland/forest)
Mosaic vegetation (grassland/shrubland/forest) / cropland
Broadleaved evergreen or semi-deciduous forest
Closed broadleaved deciduous forest
Open broadleaved deciduous forest/woodland
Mosaic forest or shrubland / grassland
(Broadleaved or needleleaved, evergreen or deciduous) shrubland
Herbaceous vegetation (grassland, savannas or lichens/mosses)
Broadleaved forest regularly flooded (semi-permanently or temporarily)

Table 3: Land categories in Figure 10.

Conclusions

This paper and the **mapmisc** package aim to contribute to and advance the suite of tools available to the growing community of R users performing advanced statistical analyses of spatial data. The first objective of **mapmisc** is removing barriers to using a map projection which is appropriate for the problem at hand, even when a non-standard projection optimised for a particular study region is most appropriate. A second objective is the provision of tools which simplify the creation and use of colour scales and legends. The way in which **mapmisc** seeks to accomplish these goals is by automating many of the tasks involved, with obtaining and re-projecting map tiles or creating colour scales with transparency being two examples. New functionality provided by **mapmisc** to R users includes the creation of optimised map projections and the creation of inset maps and scale bars for data in a rotated projection.

The scope of **mapmisc** has been kept deliberately narrow. Maps are static rather than interactive, base graphics are used, all spatial data types used are from **sp** or **raster**, and functions have been kept simple with few arguments. In contrast to `spplot` from package **sp** and **ggplot2** (Wickham, 2009), maps made with **mapmisc** are produced with a sequence of independent function calls with each function performing a very specific task. This approach was originally intended to benefit students and non-specialists, and the package grew out of code originally provided to students in an undergraduate course. Many of the tools in **mapmisc** can, however, be used with **ggplot2**, **leaflet** (Cheng and Xie, 2016) or other advanced graphical packages. No new object classes have been created by **mapmisc**, any package compatible with **raster** can use background maps from `openmap` and a "CRS" object

provided by `omerc` or `moll` can be used with any package that calls **rgdal** for transformations. Colours and break points from `colourScale` can be used with `spplot`, and `legendBreaks` can be used in any graphics environment where the `legend` function operates.

Additional motivations for the framework **mapmisc** employs are reproducibility and consistency when producing multiple maps. Colour scales and background maps are defined before a map is produced, and plot areas are laid out before any graphics are added. While sometimes clumsy for interactive use, **mapmisc** code fits tidily within `Sweave` or **knitr** documents and script files. Reproducibility of research results is an area where R excels, and **mapmisc** can help to simplify the creation of high quality maps in reproducible code scripts. For refining and manually polishing maps and plots, R will never replace Geographical Information Systems or graphics editing software. For most other tasks faced by a Spatial Statistician, however, the occasions when an environment other than R is required are becoming fewer in number over time.

Acknowledgements

The author holds a Discovery Grant from the Natural Sciences and Engineering Council of Canada.

Attributions for background maps

Figures 2a, 8 and all inset maps: © OpenStreetMap contributors. Data by OpenStreetMap available under the Open Database License, cartography is licensed as CC BY-SA.

Figure 2b, 9b: Tiles courtesy of MapQuest, portions courtesy NASA/JPL-Caltech and U.S. Depart. of Agriculture, Farm Service Agency.

Figure 7 Cartography by The Canada Base Map – Transportation (CBMT) web mapping services of the Earth Sciences Sector (ESS) at Natural Resources Canada (NRCan) licensed as the Open Government Licence – Canada.

Figure 10: Map tiles by Stamen Design under CC BY 3.0. Data by OpenStreetMap available under the CC BY-SA.

Figure 6: Map tiles by CartoDB under CC BY 3.0. Data by OpenStreetMap available under the Open Database License.

Figure 11: The Appendix is © OpenStreetMap contributors. Data by OpenStreetMap available under the Open Database License, cartography is licensed as CC BY-SA. with the exceptions below.

mapquest : Tiles courtesy of MapQuest. Data by OpenStreetMap available under the Open Database License.

mapquest-sat : Tiles courtesy of MapQuest, portions courtesy NASA/JPL-Caltech and U.S. Depart. of Agriculture, Farm Service Agency.

mapquest-labels : Tiles courtesy of MapQuest. Data by OpenStreetMap available under the Open Database License.

maptoolkit : © Toursprung GmbH Data by OpenStreetMap available under the Open Database License.

humanitarian : © OpenStreetMap contributors. Data by OpenStreetMap available under the Open Database License, cartography by Humanitarian OSM team is licensed as CC BY-SA.

cartodb : Map tiles by CartoDB under CC BY 3.0. Data by OpenStreetMap available under the Open Database License.

cartodb-dark : Map tiles by CartoDB under CC BY 3.0. Data by OpenStreetMap available under the Open Database License.

stamen-toner : Map tiles by Stamen Design under CC BY 3.0. Data by OpenStreetMap available under the Open Database License.

stamen-watercolor : Map tiles by Stamen Design under CC BY 3.0. Data by OpenStreetMap available under the CC BY-SA.

Bibliography

H. Bengtsson. *R.utils: Various Programming Utilities*, 2016. URL `https://CRAN.R-project.org/package=R.utils`. R package version 2.3.0. [p64]

R. Bivand. *classInt: Choose Univariate Class Intervals*, 2015. URL https://CRAN.R-project.org/package=classInt. R package version 0.1-23. [p64]

R. Bivand and N. Lewin-Koh. *maptools: Tools for Reading and Handling Spatial Objects*, 2016. URL https://CRAN.R-project.org/package=maptools. R package version 0.8-39. [p64]

R. Bivand and C. Rundel. *rgeos: Interface to Geometry Engine – Open Source (GEOS)*, 2016. URL https://CRAN.R-project.org/package=rgeos. R package version 0.3-19. [p64]

R. Bivand, T. Keitt, and B. Rowlingson. *rgdal: Bindings for the Geospatial Data Abstraction Library*, 2016. URL https://CRAN.R-project.org/package=rgdal. R package version 1.1-8. [p64]

R. S. Bivand, E. Pebesma, and V. Gomez-Rubio. *Applied Spatial Data Analysis with R*. Springer, New York, 2nd edition, 2013. URL http://www.asdar-book.org/. [p65]

P. Brown. *mapmisc: Utilities for Producing Maps*, 2016. URL https://CRAN.R-project.org/package=mapmisc. R package version 1.5.0. [p64]

P. E. Brown. Model-based geostatistics the easy way. *Journal of Statistical Software*, 63(12):1–24, 2015. URL http://www.jstatsoft.org/v63/i12/. [p66, 75]

J. Cheng and Y. Xie. *leaflet: Create Interactive Web Maps with the JavaScript "Leaflet" Library*, 2016. URL https://CRAN.R-project.org/package=leaflet. R package version 1.0.1. [p86]

R. J. Hijmans. *geosphere: Spherical Trigonometry*, 2015a. URL https://CRAN.R-project.org/package=geosphere. R package version 1.5-1. [p64]

R. J. Hijmans. *raster: Geographic Data Analysis and Modeling*, 2015b. URL https://CRAN.R-project.org/package=raster. R package version 2.5-2. [p64]

R. J. Hijmans, S. Phillips, J. Leathwick, and J. Elith. *dismo: Species Distribution Modeling*, 2016. URL https://CRAN.R-project.org/package=dismo. R package version 1.0-15. [p64]

G. F. Jenks and F. C. Caspall. Error on choroplethic maps: Definition, measurement, reduction. *The Annals of the Association of American Geographers*, 61(2):217–244, 1971. [p80]

R. Munroe. Map projections. xkcd Web Comic, 2011. URL http://xkcd.com/977. [Online; accessed 11-March-2015]. [p72]

E. Neuwirth. *RColorBrewer: ColorBrewer Palettes*, 2014. URL https://CRAN.R-project.org/package=RColorBrewer. R package version 1.1-2. [p64]

E. J. Pebesma and R. S. Bivand. Classes and methods for spatial data in R. *R News*, 5(2):9–13, November 2005. URL http://CRAN.R-project.org/doc/Rnews/. [p64, 80]

J. P. Snyder. Map projections – A working manual. Professional Paper 1395, US Geologic Survey, Washington, DC, 1987. URL http://pubs.usgs.gov/pp/1395/report.pdf. [p66]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009. URL http://ggplot2.org. [p86]

Wikipedia. Great-circle distance – Wikipedia, The Free Encyclopedia, 2015a. URL https://en.wikipedia.org/w/index.php?title=Great-circle_distance&oldid=688488703. [Online; accessed 3-November-2015]. [p66]

Wikipedia. Mollweide projection – Wikipedia, The Free Encyclopedia, 2015b. URL https://en.wikipedia.org/w/index.php?title=Mollweide_projection&oldid=689173007. [Online; accessed 25-November-2015]. [p72]

Wikipedia. Web Mercator – Wikipedia, The Free Encyclopedia, 2016. URL https://en.wikipedia.org/w/index.php?title=Web_Mercator&oldid=714750105. [Online; accessed 19-May-2016]. [p70]

Y. Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition, 2015. URL http://yihui.name/knitr/. [p69]

Patrick Brown
Cancer Care Ontario
620 University Ave
Toronto, ON M5G 2L7 Canada
patrick.brown@utoronto.ca

Additional code

European fertility

URL's for web sites where data are obtained:

```
nutsUrl <- "http://ec.europa.eu/eurostat/cache/GISCO/geodatafiles/NUTS_2010_60M_SH.zip"
nutsFile <- basename(nutsUrl)
```

Download and read in boundary file:

```
if (!file.exists(nutsFile)) {
    download.file(nutsUrl, nutsFile, method = "curl")
}
unzip(nutsFile)
nutsShp <- grep("RG_60M_2010.shp$", unzip(nutsFile, list = TRUE)$Name, value = TRUE)
euroNuts <- shapefile(nutsShp)
```

The fertility data are retrieved as a gzipped tab-separated text file, which the **R.utils** package is able to decompress. The code below will download a copy of this file from the author's web server if the EUROSTAT download fails.

```
fertUrl <- file.path("http://ec.europa.eu/eurostat/estat-navtree-portlet-prod",
    "BulkDownloadListing?sort=1&file=data%2Fdemo_r_frate2.tsv.gz")
fertFileGz <- "demo_r_frate2.tsv.gz"
fertFile <- gsub(".gz$", "", fertFileGz)
if (!file.exists(fertFileGz)) {
    download.file(fertUrl, fertFileGz, method = "curl")
}
R.utils::gunzip(fertFileGz, overwrite = file.exists(fertFile), remove = FALSE)
euroDat <- read.table(fertFile, header = TRUE, stringsAsFactors = FALSE, sep = "\t",
    na.strings = ": ")
euroDat <- euroDat[grep("^TOTAL", euroDat[, 1]), ]
euroDat$timegeo <- gsub("^TOTAL,", "", euroDat[, 1])
```

Merge the fertility and polygon data. Warning messages that not all rows of the fertility table can be matched to polygons can be ignored.

```
euroF <- sp::merge(euroNuts, euroDat, all.x = FALSE, by.x = "NUTS_ID",
    by.y = "timegeo")
```

Exclude some of the outlying parts of the EU:

```
euroF <- raster::crop(euroF, extent(-25, 180, 33, 90))
```

Canada

Transform the Great Circle:

```
gcircleT <- mapply(spTransform, CRSobj = crsList, MoreArgs = list(x = gcircle))
```

Obtain background maps:

```
mapT <- mapply(openmap, crs = crsList, MoreArgs = list(path = "nrcan", x = cities,
    buffer = 3))
```

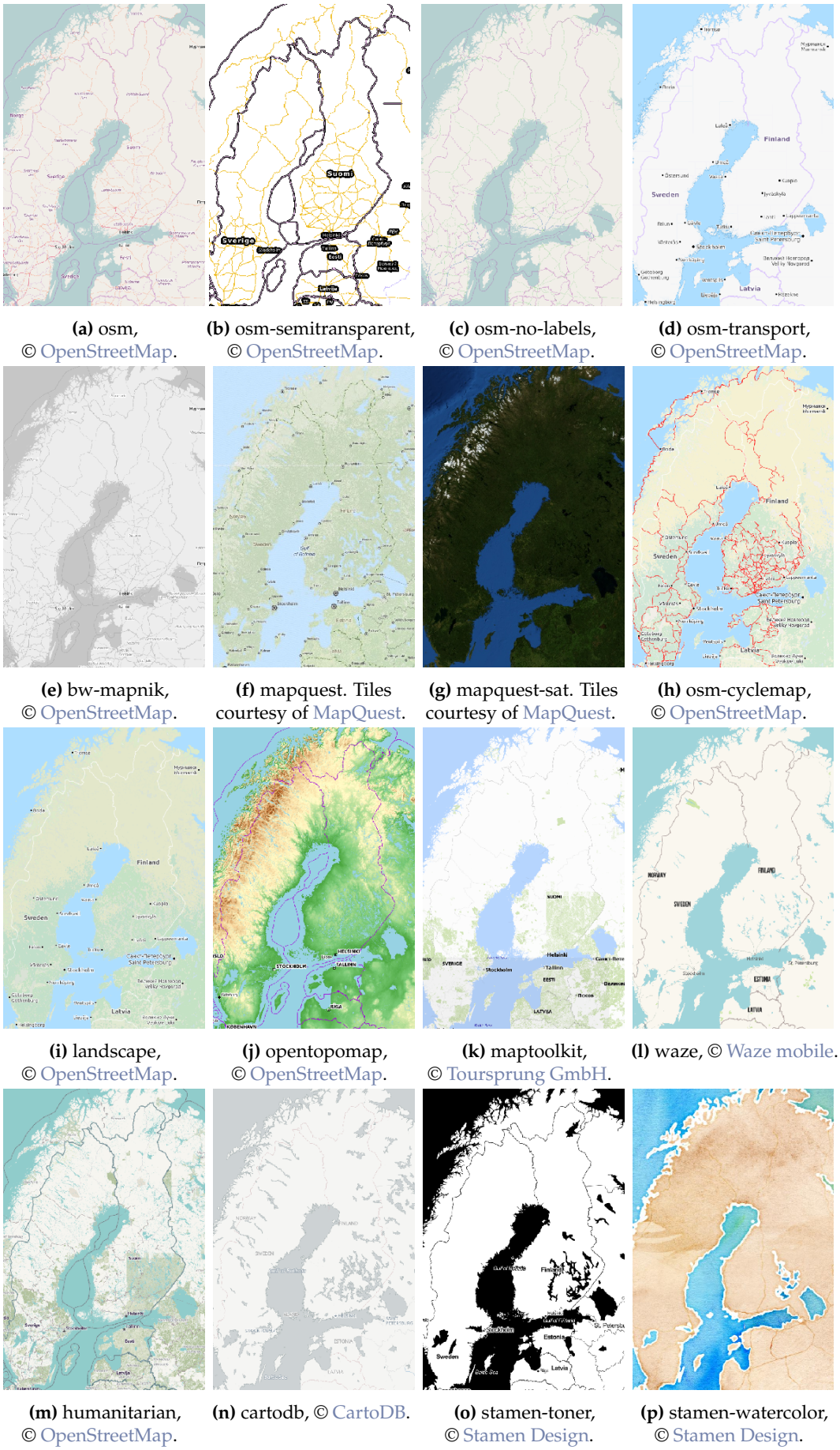
Create maps:

```
for (D in names(crsList)) {
    map.new(citiesT[[D]], buffer = c(200 * 1000, 1)[1 + isLonLat(citiesT[[D]])])
    plotRGB(mapT[[D]], add = TRUE)
    rgdal::llgridlines(citiesT[[D]], col = "orange")
    points(gcircleT[[D]], cex = 0.1, col = "blue")
    points(citiesT[[D]], col = "red", pch = "+", cex = 1.5)
    text(citiesT[[D]], labels = citiesT[[D]]$name, col = "red", pos = cities$pos,
        offset = 0.6, cex = 1.2)
    scaleBar(citiesT[[D]], "topleft", seg.len = 4, pt.cex = 0)
}
```

Additional tables and figures

col	max	palette
BrBG	11	
PiYG	11	
PRGn	11	
PuOr	11	
RdBu	11	
RdGy	11	
RdYlBu	11	
RdYlGn	11	
Spectral	11	
Accent	8	
Dark2	8	
Paired	12	
Pastel1	9	
Pastel2	8	
Set1	9	
Set2	8	
Set3	12	
Blues	9	
BuGn	9	
BuPu	9	
GnBu	9	
Greens	9	
Greys	9	
Oranges	9	
OrRd	9	
PuBu	9	
PuBuGn	9	
PuRd	9	
Purples	9	
RdPu	9	
Reds	9	
YlGn	9	
YlGnBu	9	
YlOrBr	9	
YlOrRd	9	

Table 4: Colour palettes from **RColorBrewer**, with `col` showing the character string to provide `colourScale` or `brewer.pal` and `max` giving the maximum number of colours for each palette.



(a) osm, © OpenStreetMap.

(b) osm-semitransparent, © OpenStreetMap.

(c) osm-no-labels, © OpenStreetMap.

(d) osm-transport, © OpenStreetMap.

(e) bw-mapnik, © OpenStreetMap.

(f) mapquest. Tiles courtesy of MapQuest.

(g) mapquest-sat. Tiles courtesy of MapQuest.

(h) osm-cyclemap, © OpenStreetMap.

(i) landscape, © OpenStreetMap.

(j) opentopomap, © OpenStreetMap.

(k) maptoolkit, © Toursprung GmbH.

(l) waze, © Waze mobile.

(m) humanitarian, © OpenStreetMap.

(n) cartodb, © CartoDB.

(o) stamen-toner, © Stamen Design.

(p) stamen-watercolor, © Stamen Design.

Figure 11: Selected openmap tile sets.

Variable Clustering in High-Dimensional Linear Regression: The R Package clere

by Loïc Yengo, Julien Jacques, Christophe Biernacki and Mickael Canouil

Abstract Dimension reduction is one of the biggest challenges in high-dimensional regression models. We recently introduced a new methodology based on variable clustering as a means to reduce dimensionality. We present here the R package **clere** that implements some refinements of this methodology. An overview of the package functionalities as well as examples to run an analysis are described. Numerical experiments on real data were performed to illustrate the good predictive performance of our parsimonious method compared to standard dimension reduction approaches.

Introduction

High dimensionality is increasingly ubiquitous in numerous scientific fields including genetics, economics and physics. Reducing the dimensionality is a challenge that most statistical methodologies must meet not only to remain interpretable but also to achieve reliable predictions. In linear regression models, dimension reduction techniques often correspond to variable selection methods. Approaches for variable selection are already implemented in publicly available, open-source software, e.g., the well-known R packages **glmnet** (Friedman et al., 2010) and **spikeslab** (Ishwaran et al., 2013). The R package **glmnet** implements the Elastic net methodology (Zou and Hastie, 2005), which is a generalization of both the LASSO (Tibshirani, 1996) and the ridge regression (RR; Hoerl and Kennard, 1970). The R package **spikeslab** in turn, implements the Spike and Slab methodology (Ishwaran and Rao, 2005), which is a Bayesian approach for variable selection.

Dimension reduction cannot, however, be restricted to variable selection. Indeed, the field can be extended to include approaches which aim at creating surrogate covariates that summarize the information contained in initial covariates. Since the emblematic principal component regression (PCR; Jolliffe, 1982), many other methods spread in the recent literature. As specific examples, we may refer to the OSCAR methodology (Bondell and Reich, 2008), or the PACS methodology (Sharma et al., 2013) which is a generalization of the latter approach. Those methods mainly proposed variable clustering within a regression model as a way to reduce the dimensionality. Despite their theoretical and practical appeal, implementations of those methods were often proposed only through MATLAB (The MathWorks Inc., 2014) or R scripts, limiting thus the flexibility and the computational efficiency of their use. The CLusterwise Effect REgression (CLERE) methodology (Yengo et al., 2014), was recently introduced as a novel methodology for simultaneous variable clustering and regression. The CLERE methodology is based on the assumption that each regression coefficient is an unobserved random variable sampled from a mixture of Gaussian distributions with an arbitrary number g of components. In addition, all components in the mixture are assumed to have different means $(b_1, \ldots, b_g)$ and equal variances γ^2.

In this paper, we propose two new features for the CLERE model. First, the stochastic EM (SEM) algorithm is proposed as a more computationally efficient alternative to the Monte Carlo EM (MCEM) algorithm previously introduced in Yengo et al. (2014). Secondly, the CLERE model is enhanced with the possibility of constraining the first component to have its mean equal to 0, i.e. $b_1 = 0$. This enhancement is mainly aimed at facilitating the interpretation of the model. Indeed when b_1 is set to 0, variables assigned to the cluster associated with b_1 might be considered less relevant than other variables provided γ^2 is small enough. Those two new features were implemented in the R package **clere** (Yengo and Canouil, 2015). The core of the package is a C++ program interfaced with R using the R packages **Rcpp** (Eddelbuettel and François, 2011) and **RcppEigen** (Bates and Eddelbuettel, 2013). The R package **clere** can be downloaded from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=clere`.

The outline of the present paper is the following. In the next section the definition of the model is recalled and the strategy to estimate the model parameters is explained. Afterwards, the main functionalities of the R package **clere** are presented. Real data analyses are then provided, aiming at illustrating the good predictive performances of CLERE, with noticeable parsimony ability, compared to standard dimension reduction methods. Finally, perspectives and further potential improvements of the package are discussed in the last section.

Model definition and notation

Our model is defined by the following hierarchical relationships:

$$\begin{cases} ly_i \sim \mathcal{N}\left(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}, \sigma^2\right), \\ \beta_j | \mathbf{z}_j \sim \mathcal{N}\left(\sum_{k=1}^{g} b_k z_{jk}, \gamma^2\right), \\ \mathbf{z}_j = \left(z_{j1}, \ldots, z_{jg}\right) \sim \mathcal{M}\left(1, \pi_1, \ldots, \pi_g\right), \end{cases} \tag{1}$$

where $\mathcal{N}(\mu, \sigma^2)$ is the normal distribution with mean μ and variance σ^2, $\mathcal{M}\left(1, \pi_1, \ldots, \pi_g\right)$ the one-order multinomial distribution with parameters $\boldsymbol{\pi} = \left(\pi_1, \ldots, \pi_g\right)$, where, $\forall\, k = 1, \ldots, g$ $\pi_k > 0$ and $\sum_{k=1}^{g} \pi_k = 1$, and β_0 is a constant term. For an individual $i = 1, \ldots, n$, y_i is the response and x_{ij} is an observed value for the j-th covariate. β_j is the regression coefficient associated with the j-th covariate ($j = 1, \ldots, p$), which is assumed to follow a mixture of g Gaussians. The variable $\mathbf{z}_j$ indicates from which mixture component β_j is drawn ($z_{jk} = 1$ if β_j comes from component k of the mixture, $z_{jk} = 0$ otherwise). Let's note that model (1) can be considered as a variable selection-like model by constraining the model parameter b_1 to be equal to 0. Indeed, assuming that one of the components is centered in zero means that a cluster of regression coefficients have null expectation, and thus that the corresponding variables are not significant for explaining the response variable. This functionality is available in the package.

Let $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)$, $\boldsymbol{y} = (y_1, \ldots, y_n)'$, $\boldsymbol{X} = (x_{ij})$, $\boldsymbol{Z} = (z_{jk})$, $\boldsymbol{b} = (b_1, \ldots, b_g)'$ and $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_g)'$. Moreover, $\log p(\boldsymbol{y}|\boldsymbol{X}; \boldsymbol{\theta})$ denotes the log-likelihood of model (1) assessed for the parameter vector $\boldsymbol{\theta} = \left(\beta_0, \boldsymbol{b}, \boldsymbol{\pi}, \sigma^2, \gamma^2\right)$. Model (1) can be interpreted as a Bayesian approach. However, to be fully Bayesian a prior distribution for parameter $\boldsymbol{\theta}$ would have been necessary. Instead, we proposed to estimate $\boldsymbol{\theta}$ by maximizing the (marginal) log-likelihood, $\log p(\mathbf{y}|\mathbf{X}; \boldsymbol{\theta})$. This partially Bayesian approach is referred to as *Empirical Bayes* (EB; Casella, 1985). Let $\mathcal{Z}$ be the set of $p \times g$-matrices partitioning p covariates into g groups. Those matrices are defined as

$$\boldsymbol{Z} = \left(z_{jk}\right)_{1 \leq j \leq p, 1 \leq k \leq g} \in \mathcal{Z} \Leftrightarrow \forall j = 1, \ldots, p \begin{cases} \exists!\ k \text{ such as } z_{jk} = 1 \\ \text{For all } k' \neq k\ z_{jk'} = 0. \end{cases}$$

The log-likelihood $\log p(\boldsymbol{y}|\boldsymbol{X}; \boldsymbol{\theta})$ is defined as

$$\log p(\boldsymbol{y}|\boldsymbol{X}; \boldsymbol{\theta}) = \log \left[\sum_{\boldsymbol{Z} \in \mathcal{Z}} \int_{\mathbb{R}^p} p(\boldsymbol{y}, \boldsymbol{\beta}, \boldsymbol{Z}|\boldsymbol{X}; \boldsymbol{\theta}) \mathrm{d}\boldsymbol{\beta} \right].$$

Since it requires integrating over $\mathcal{Z}$ with cardinality g^p, evaluating the likelihood becomes rapidly computationally unaffordable.

Nonetheless, maximum likelihood estimation is still achievable using the expectation maximization (EM) algorithm (Dempster et al., 1977). The latter algorithm is an iterative method which starts with an initial estimate of the parameter and updates this estimate until convergence. Each iteration of the algorithm consists of two steps, denoted as the *E* and the *M steps*. At each iteration d of the algorithm, the *E step* consists in calculating the expectation of the log-likelihood of the complete data (observed + unobserved) with respect to $p(\boldsymbol{\beta}, \boldsymbol{Z}|\boldsymbol{y}, \boldsymbol{X}; \boldsymbol{\theta}^{(d)})$, the conditional distribution of the unobserved data given the observed data, and the value of the parameter at the current iteration, $\boldsymbol{\theta}^{(d)}$. This expectation, often denoted as $Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(d)})$ is then maximized with respect to $\boldsymbol{\theta}$ at the *M step*.

In model (1), the *E step* is analytically intractable. A broad literature devoted to intractable *E steps* recommends the use of a stochastic approximation of $Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(d)})$ through Monte Carlo (MC) simulations (Wei and Tanner, 1990; Levine and Casella, 2001). This approach is referred to as the MCEM algorithm. Besides, mean-field-type approximations are also proposed (Govaert and Nadif, 2008; Mariadassou et al., 2010). Despite their computational appeal, the latter approximations do not generally ensure convergence to the maximum likelihood (Gunawardana and Byrne, 2005). Alternatively, the SEM algorithm (Celeux et al., 1996) was introduced as a stochastic version of the EM algorithm. In this algorithm, the *E step* is replaced with a simulation step (*S step*) that consists in generating a complete sample by simulating the unobserved data using $p(\boldsymbol{\beta}, \boldsymbol{Z}|\boldsymbol{y}, \boldsymbol{X}; \boldsymbol{\theta}^{(d)})$ providing thus a sample $(\boldsymbol{\beta}^{(d)}, \boldsymbol{Z}^{(d)})$. Note that the Monte Carlo algorithm we use to perform this simulation is the Gibbs sampler. After the *S step* follows the *M step* which consists in maximizing $p(\boldsymbol{\beta}^{(d)}, \boldsymbol{Z}^{(d)}|\boldsymbol{y}, \boldsymbol{X}; \boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$. Alternating those two steps generates a sequence $\left(\boldsymbol{\theta}^{(d)}\right)$, which is a Markov chain whose stationary distribution (when it exists) concentrates around the local maxima of the likelihood.

Estimation and model selection

In this section, two algorithms for model inference are presented: the Monte-Carlo Expectation Maximization (MCEM) algorithm and the Stochastic Expectation Maximization (SEM) algorithm. The section starts with the initialization strategy common to both algorithms and continues with the detailed description of each algorithm. Then, model selection (for choosing g) and variable selection are discussed.

Initialization

The two algorithms presented in this section are initialized using a primary estimate ${\beta_j}^{(0)}$ of each β_j. The latter can be chosen either at random, or obtained from univariate regression coefficients or penalized approaches like LASSO and ridge regression. For large SEM or MCEM chains, initialization is not a critical issue. The choice of the initialization strategy is therefore made to speed up the convergence of the chains. A Gaussian mixture model with g component(s) is then fitted using $\boldsymbol{\beta}^{(0)} = \left(\beta_1^{(0)}, \ldots, \beta_p^{(0)}\right)$ as observed data to produce starting values $\mathbf{b}^{(0)}$, $\boldsymbol{\pi}^{(0)}$ and ${\gamma^2}^{(0)}$ respectively for parameters $\mathbf{b}$, $\boldsymbol{\pi}$ and γ^2. Using maximum a posteriori (MAP) clustering, an initial partition $\mathbf{Z}^{(0)} = \left(z_{jk}^{(0)}\right) \in \mathcal{Z}$ is obtained as

$$\forall j \in \{1, \ldots, p\},\ z_{jk}^{(0)} = \begin{cases} 1 & \text{if } k = \arg\min_{k' \in \{1,\ldots,g\}} \left({\beta_j}^{(0)} - b_{k'}^{(0)}\right)^2, \\ 0 & \text{otherwise.} \end{cases}$$

β_0 and σ^2 are initialized using $\boldsymbol{\beta}^{(0)}$ as follows:

$$\beta_0^{(0)} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p}\beta_j^{(0)}x_{ij}\right) \text{ and } {\sigma^2}^{(0)} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta_0^{(0)} - \sum_{j=1}^{p}\beta_j^{(0)}x_{ij}\right)^2.$$

MCEM algorithm

The stochastic approximation of the E step

Suppose at iteration d of the algorithm that we have $\left\{\left(\boldsymbol{\beta}^{(1,d)}, \boldsymbol{Z}^{(1,d)}\right), \ldots, \left(\boldsymbol{\beta}^{(M,d)}, \boldsymbol{Z}^{(M,d)}\right)\right\}$, M samples from $p\left(\boldsymbol{\beta}, \boldsymbol{Z}|\boldsymbol{y}, \boldsymbol{X}; \boldsymbol{\theta}^{(d)}\right)$. Then the MC approximation of the *E step* can be written as

$$Q\left(\boldsymbol{\theta}|\boldsymbol{\theta}^{(d)}\right) = \mathbb{E}\left[\log p(\mathbf{y}, \boldsymbol{\beta}, \mathbf{Z}|\mathbf{X}; \boldsymbol{\theta}^{(d)})|\mathbf{y}, \mathbf{X}; \boldsymbol{\theta}^{(d)}\right] \approx \frac{1}{M}\sum_{m=1}^{M}\log p(\mathbf{y}, \boldsymbol{\beta}^{(m,d)}, \mathbf{Z}^{(m,d)}|\mathbf{X}; \boldsymbol{\theta}^{(d)}).$$

Sampling from $p\left(\boldsymbol{\beta}, \boldsymbol{Z}|\boldsymbol{y}, \boldsymbol{X}; \boldsymbol{\theta}^{(d)}\right)$ is not straightforward. However, we can use a Gibbs sampling scheme to simulate unobserved data, taking advantage of $p\left(\boldsymbol{\beta}|\boldsymbol{Z}, \boldsymbol{y}, \boldsymbol{X}; \boldsymbol{\theta}^{(d)}\right)$ and $p\left(\boldsymbol{Z}|\boldsymbol{\beta}, \boldsymbol{y}, \boldsymbol{X}; \boldsymbol{\theta}^{(d)}\right)$ from which it is easy to simulate. These distributions, i.e., Gaussian and multinomial, respectively, are described below in Equations (2) and (3).

$$\begin{cases} \boldsymbol{\beta}|\boldsymbol{Z}, \boldsymbol{y}, \boldsymbol{X}; \boldsymbol{\theta}^{(d)} \sim \mathcal{N}\left(\boldsymbol{\mu}^{(d)}, \boldsymbol{\Sigma}^{(d)}\right), \\ \boldsymbol{\mu}^{(d)} = \left[\boldsymbol{X'X} + \frac{{\sigma^2}^{(d)}}{{\gamma^2}^{(d)}}\mathrm{I}_p\right]^{-1}\boldsymbol{X'}\left(\mathbf{y} - \beta_0^{(d)}\mathbf{1}_p\right) + \frac{{\sigma^2}^{(d)}}{{\gamma^2}^{(d)}}\left[\boldsymbol{X'X} + \frac{{\sigma^2}^{(d)}}{{\gamma^2}^{(d)}}\mathrm{I}_p\right]^{-1}\boldsymbol{Zb}^{(d)}, \\ \boldsymbol{\Sigma}^{(d)} = {\sigma^2}^{(d)}\left[\boldsymbol{X'X} + \frac{{\sigma^2}^{(d)}}{{\gamma^2}^{(d)}}\mathrm{I}_p\right]^{-1}, \end{cases} \tag{2}$$

and, noting that $p\left(\boldsymbol{Z}|\boldsymbol{\beta}, \boldsymbol{y}, \boldsymbol{X}; \boldsymbol{\theta}^{(d)}\right)$ does not depend on $\boldsymbol{X}$ nor $\boldsymbol{y}$,

$$p\left(z_{jk} = 1|\boldsymbol{\beta}; \boldsymbol{\theta}^{(d)}\right) \propto \pi_k^{(d)}\exp\left(-\frac{\left(\beta_j - b_k^{(d)}\right)^2}{2{\gamma^2}^{(d)}}\right). \tag{3}$$

In Equation (2), I_p and $\mathbf{1}_p$ stand for the identity matrix with dimension p and the vector of $\mathbb{R}^p$ where all elements are equal to 1. To efficiently sample from $p\left(\boldsymbol{\beta}|\boldsymbol{Z},\boldsymbol{y},\boldsymbol{X};\boldsymbol{\theta}^{(d)}\right)$ a preliminary singular vector decomposition of matrix $\boldsymbol{X}$ is necessary. Once this decomposition is performed the overall complexity of the approximate *E step* is $\mathcal{O}\left[M(p^2+pg)\right]$.

The M step

Using the M draws obtained by Gibbs sampling at iteration d, the *M step* is straightforward as detailed in Equations (4) to (8). The overall computational complexity of that step is $\mathcal{O}\left(Mpg\right)$.

$$\pi_k^{(d+1)} = \frac{1}{Mp}\sum_{m=1}^{M}\sum_{j=1}^{p} z_{jk}^{(m,d)}, \tag{4}$$

$$b_k^{(d+1)} = \frac{1}{Mp\pi_k^{(d+1)}}\sum_{m=1}^{M}\sum_{j=1}^{p} z_{jk}^{(m,d)}\beta_j^{(m,d)}, \tag{5}$$

$$\gamma^{2(d+1)} = \frac{1}{Mp}\sum_{m=1}^{M}\sum_{j=1}^{p}\sum_{k=1}^{g} z_{jk}^{(m,d)}\left(\beta_j^{(m,d)} - b_k^{(d+1)}\right)^2, \tag{6}$$

$$\beta_0^{(d+1)} = \frac{1}{n}\sum_{i=1}^{n}\left[y_i - \sum_{j=1}^{p}\left(\frac{1}{M}\sum_{m=1}^{M}\beta_j^{(m,d)}\right)x_{ij}\right], \tag{7}$$

$$\sigma^{2(d+1)} = \frac{1}{nM}\sum_{m=1}^{M}\sum_{i=1}^{n}\left(y_i - \beta_0^{(d+1)} - \sum_{j=1}^{p}\beta_j^{(m,d)}x_{ij}\right)^2. \tag{8}$$

SEM algorithm

In most situations, the SEM algorithm can be considered as a special case of the MCEM algorithm (Celeux et al., 1996), obtained by setting $M = 1$. In model (1), such a direct derivation leads to an algorithm where the computational complexity remains quadratic with respect to p. To reduce that complexity, we propose a SEM algorithm based on the integrated complete data likelihood $p(\boldsymbol{y},\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta})$ rather than $p(\boldsymbol{y},\boldsymbol{\beta},\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta})$. A closed form of $p(\boldsymbol{y},\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta})$ is available and given in the following.

Closed form of the integrated complete data likelihood

Let the SVD decomposition of matrix $\boldsymbol{X}$ be $\boldsymbol{USV}'$, where $\boldsymbol{U}$ and $\boldsymbol{V}$ are respectively $n \times n$ and $p \times p$ orthogonal matrices, and $\boldsymbol{S}$ is a $n \times p$ rectangular diagonal matrix where the diagonal terms are the eigenvalues $\left(\lambda_1^2,\ldots,\lambda_n^2\right)$ of matrix $\boldsymbol{XX}'$. We now define $\boldsymbol{X}^u = \boldsymbol{U}'\boldsymbol{X}$ and $\boldsymbol{y}^u = \boldsymbol{U}'\boldsymbol{y}$. Let $\boldsymbol{M}$ be the $n \times (g+1)$ matrix where the first column is made of 1's and where the additional columns are those of matrix $\boldsymbol{X}^u\boldsymbol{Z}$. Let also $\boldsymbol{t} = (\beta_0, \boldsymbol{b}) \in \mathbb{R}^{(g+1)}$ and $\boldsymbol{R}$ be a $n \times n$ diagonal matrix where the i-th diagonal term equals $\sigma^2 + \gamma^2\lambda_i^2$. With these notations we can express the complete data likelihood integrated over $\boldsymbol{\beta}$ as

$$\log p\left(\boldsymbol{y},\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}\right) = -\frac{n}{2}\log\left(2\pi\right) - \frac{1}{2}\sum_{i=1}^{n}\log\left(\sigma^2+\gamma^2\lambda_i^2\right) - \frac{1}{2}\left(\boldsymbol{y}^u - \boldsymbol{Mt}\right)'\boldsymbol{R}^{-1}\left(\boldsymbol{y}^u - \boldsymbol{Mt}\right) + \sum_{j=1}^{p}\sum_{k=1}^{g} z_{jk}\log\pi_k. \tag{9}$$

Simulation step

To sample from $p\left(\boldsymbol{Z}|\boldsymbol{y},\boldsymbol{X};\boldsymbol{\theta}\right)$ we use a Gibbs sampling strategy based on the conditional distributions $p\left(\boldsymbol{z}_j|\boldsymbol{y},\boldsymbol{Z}^{-j},\boldsymbol{X};\boldsymbol{\theta}\right)$, $\boldsymbol{Z}^{-j}$ denoting the set of cluster membership indicators for all covariates but the j-th. Let $\boldsymbol{w}^{-j} = \left(w_1^{-j},\ldots,w_n^{-j}\right)'$, where $w_i^{-j} = y_i^u - \beta_0 - \sum_{l\neq j}\sum_{k=1}^{g} z_{lk}x_{il}^u b_k$. The conditional distribution

$p(z_{jk} = 1|\boldsymbol{Z}^{-j}, \boldsymbol{y}, \boldsymbol{X}; \boldsymbol{\theta})$ can be written as

$$p(z_{jk} = 1|\boldsymbol{Z}^{-j}, \boldsymbol{y}, \boldsymbol{X}; \boldsymbol{\theta}) \propto \pi_k \exp\left[-\frac{b_k^2}{2}\left(\boldsymbol{x}_j^u\right)' \boldsymbol{R}^{-1}\boldsymbol{x}_j^u + b_k\left(\boldsymbol{w}^{-j}\right)' \boldsymbol{R}^{-1}\boldsymbol{x}_j^u\right], \tag{10}$$

where $\boldsymbol{x}_j^u$ is the j-th column of $\boldsymbol{X}^u$. In the classical SEM algorithm, convergence to $p\left(\boldsymbol{Z}|\boldsymbol{y}, \boldsymbol{X}; \boldsymbol{\theta}\right)$ should be reached before updating $\boldsymbol{\theta}$. However, a valid inference can still be ensured in settings when $\boldsymbol{\theta}$ is updated only after one or few Gibbs iterations. These approaches are referred to as SEM-Gibbs algorithm (Biernacki and Jacques, 2013). The overall computational complexity of the simulation step is $\mathcal{O}\left(npg\right)$, i.e., it is linear in p and not quadratic any more, in contrast to the previous MCEM.

To improve the mixing of the generated Markov chain, we start the simulation step at each iteration by creating a random permutation of $\{1, \ldots, p\}$. Then, according to the order defined by that permutation, we update each z_{jk} using $p(z_{jk} = 1|\boldsymbol{Z}^{-j}, \boldsymbol{y}, \boldsymbol{X}; \boldsymbol{\theta})$.

Maximization step

$\log p\left(\boldsymbol{y}, \boldsymbol{Z}|\boldsymbol{X}; \boldsymbol{\theta}\right)$ corresponds to the marginal log-likelihood of a linear mixed model (Searle et al., 1992), which can be written as

$$\boldsymbol{y}^u = \boldsymbol{M}\boldsymbol{t} + \boldsymbol{\lambda}\boldsymbol{v} + \boldsymbol{\varepsilon} \tag{11}$$

where $\boldsymbol{v}$ is an unobserved random vector such as $\boldsymbol{v} \sim \mathcal{N}\left(0, \gamma^2 \mathrm{I}_n\right)$, $\boldsymbol{\varepsilon} \sim \mathcal{N}\left(0, \sigma^2 \mathrm{I}_n\right)$ and $\boldsymbol{\lambda} = \mathrm{diag}\left(\lambda_1, \ldots, \lambda_n\right)$. The estimation of the parameters of model (11) can be performed using the EM algorithm, as in Searle et al. (1992). We adapt below the EM equations defined in Searle et al. (1992), using our notations. At iteration s of the internal EM algorithm, we define $\boldsymbol{R}^{(s)} = \sigma^{2^{(s)}}\boldsymbol{I}_n + \gamma^{2^{(s)}}\boldsymbol{\lambda}'\boldsymbol{\lambda}$. The detailed *internal E* and *M steps* are given below.

Internal E step

$$\begin{aligned}
v_\sigma^{(s)} &= \mathbb{E}\left[\left(\boldsymbol{y}^u - \boldsymbol{M}\boldsymbol{t}^{(s)} - \boldsymbol{\lambda}\boldsymbol{v}\right)'\left(\boldsymbol{y}^u - \boldsymbol{M}\boldsymbol{t}^{(s)} - \boldsymbol{\lambda}\boldsymbol{v}\right)|\boldsymbol{y}^u\right] \\
&= \sigma^{4^{(s)}}\left(\boldsymbol{y}^u - \boldsymbol{M}\boldsymbol{t}^{(s)}\right)'\boldsymbol{R}^{(s)}\boldsymbol{R}^{(s)}\left(\boldsymbol{y}^u - \boldsymbol{M}\boldsymbol{t}^{(s)}\right) + n \times \sigma^{2^{(s)}} - \sigma^{4^{(s)}}\sum_{i=1}^{n}\frac{1}{\sigma^{2^{(s)}} + \gamma^{2^{(s)}}\lambda_i^2}. \\
v_\gamma^{(s)} &= \mathbb{E}\left[\boldsymbol{v}'\boldsymbol{v}|\boldsymbol{y}^u\right] \\
&= \gamma^{4^{(s)}}\left(\boldsymbol{y}^u - \boldsymbol{M}\boldsymbol{t}^{(s)}\right)'\boldsymbol{R}^{(s)}\boldsymbol{\lambda}'\boldsymbol{\lambda}\boldsymbol{R}^{(s)}\left(\boldsymbol{y}^u - \boldsymbol{M}\boldsymbol{t}^{(s)}\right) + n \times \gamma^{2^{(s)}} - \gamma^{4^{(s)}}\sum_{i=1}^{n}\frac{\lambda_i^2}{\sigma^{2^{(s)}} + \gamma^{2^{(s)}}\lambda_i^2}. \\
\boldsymbol{h}^{(s)} &= \mathbb{E}\left[\boldsymbol{y}^u - \boldsymbol{\lambda}\boldsymbol{v}|\boldsymbol{y}^u\right] = \boldsymbol{M}\boldsymbol{t}^{(s)} + \sigma^{2^{(s)}}\{R^{(s)}\}^{-1}\left(\boldsymbol{y}^u - \boldsymbol{M}\boldsymbol{t}^{(s)}\right).
\end{aligned}$$

Internal M step

$$\begin{aligned}
\sigma^{2^{(s+1)}} &= v_\sigma^{(s)}/n, \\
\gamma^{2^{(s+1)}} &= v_\gamma^{(s)}/n, \\
\boldsymbol{t}^{(s+1)} &= \left[\boldsymbol{M}'\boldsymbol{M}\right]^{-1}\boldsymbol{M}'\boldsymbol{h}^{(s)}.
\end{aligned}$$

Given a non-negative user-specified threshold δ and a maximum number $N_{\max}$ of iterations, *Internal E* and *M steps* are alternated until

$$|\log p\left(\boldsymbol{y}, \boldsymbol{Z}|\boldsymbol{X}; \boldsymbol{\theta}^{(s)}\right) - \log p\left(\boldsymbol{y}, \boldsymbol{Z}|\boldsymbol{X}; \boldsymbol{\theta}^{(s+1)}\right)| < \delta \text{ or } s = N_{\max}.$$

The computational complexity of the *M step* is $\mathcal{O}\left(g^3 + ngN_{max}\right)$, thus not involving p.

Attracting and absorbing states

- *Absorbing states*. The SEM algorithm described above defines a Markov chain where the stationary distribution is concentrated around values of $\boldsymbol{\theta}$ corresponding to local maxima of the likelihood function. This chain has absorbing states in values of $\boldsymbol{\theta}$ such as $\sigma^2 = 0$ or $\gamma^2 = 0$. In fact, the *internal M step* reveals that updated values for σ^2 and γ^2 are proportional to previous values of those parameters.

- *Attracting states*. We empirically observed that attraction around $\sigma^2 = 0$ was quite frequent when using the MCEM algorithm, especially when $p > n$ and when the number M of draws was small. We therefore advocate to use at least 5 draws ($M \geq 5$ using option `nsamp` in the function `fitClere`).

Model selection

Once the MLE $\widehat{\boldsymbol{\theta}}$ is calculated (using one of the algorithms), the maximum log-likelihood and the posterior clustering matrix $\mathbb{E}\left[\boldsymbol{Z}|\boldsymbol{y}, \boldsymbol{X}; \widehat{\boldsymbol{\theta}}\right]$ are approximated using MC simulations based on Equations (9) and (10). The approximate maximum log-likelihood $\widehat{l}$, is then utilized to calculate AIC (Akaike, 1974) and BIC (Schwarz, 1978) for model selection. In model (1), those criteria can be written as

$$\begin{aligned} \text{BIC} &= -2\widehat{l} + 2(g+1)\log(n), \\ \text{AIC} &= -2\widehat{l} + 4(g+1). \end{aligned}$$

An additional criterion for model selection, namely the ICL criterion (Biernacki et al., 2000) is also implemented in the R package **clere**. The latter criterion can be written as

$$\text{ICL} = \text{BIC} - \sum_{j=1}^{p}\sum_{k=1}^{g} \pi_{jk}\log(\pi_{jk}), \tag{12}$$

where $\pi_{jk} = \mathbb{E}\left[z_{jk}|\boldsymbol{y}, \boldsymbol{X}; \widehat{\boldsymbol{\theta}}\right]$.

Interpretation of the special group of variables associated with $b_1 = 0$

The constraint $b_1 = 0$ is mainly driven by an interpretation purpose. The meaning of this group depends on both the total number g of groups and the estimated value of parameter γ^2. In fact, when $g > 1$ and γ^2 is small, covariates assigned to that group are likely less relevant to explain the response. Determining whether γ^2 is small enough is not straightforward. However, when this property holds, we may expect the groups of covariates to be separated. This would for example translate in the posterior probabilities π_{j1} being larger than 0.7. In addition to the benefit in interpretation, the constraint $b_1 = 0$, reduces the number of parameters to be estimated and consequently the variance of the predictions performed using the model.

Package functionalities

The R package **clere** mainly implements a function for parameter estimation and model selection: the function `fitClere()`. Four additional methods are also implemented in the package: for graphical representation, `plot()`; summarizing the results, `summary()`; for getting the predicted clusters of variables, `clusters()`; and for making predictions from new design matrices, `predict()`. Examples of calls to the functions presented in this section are given in the next section.

The main function `fitClere()`

The main function `fitClere()` has only three mandatory arguments: the vector of response `y` (size n), the matrix of explanatory variables `x` (size $n \times p$) and the number g of groups of regression coefficients which is expected. The optional parameter `analysis`, when it takes the value `"aic"`, `"bic"` or `"icl"`, allows to test all the possible number of groups between 1 and g. The choice between the two proposed algorithms is possible thanks to the parameter `algorithm`, but we encourage the users to use the default value, the SEM algorithm, which generally over-performs the MCEM algorithm (see the first experiment of the next section).

Several other parameters allow to tune the different numbers of iterations of the estimation algorithm. In general, the higher are these parameter values, the better is the quality of the estimation but the heavier is also the computing time. What we advice is to use the default values, and to graphically check the quality of the estimation with plots as in Figure 1: If the values of the model parameters are quite stable for a sufficient large part of the iterations, this indicates that the results are ok. If the stability is not reached sufficiently early before the end of the iterations, a higher number of iterations should be chosen.

Finally, among the remaining parameters (note that the complete list can be obtained with `help("fitClere")`), two are especially important: `parallel` allows to run parallel computations

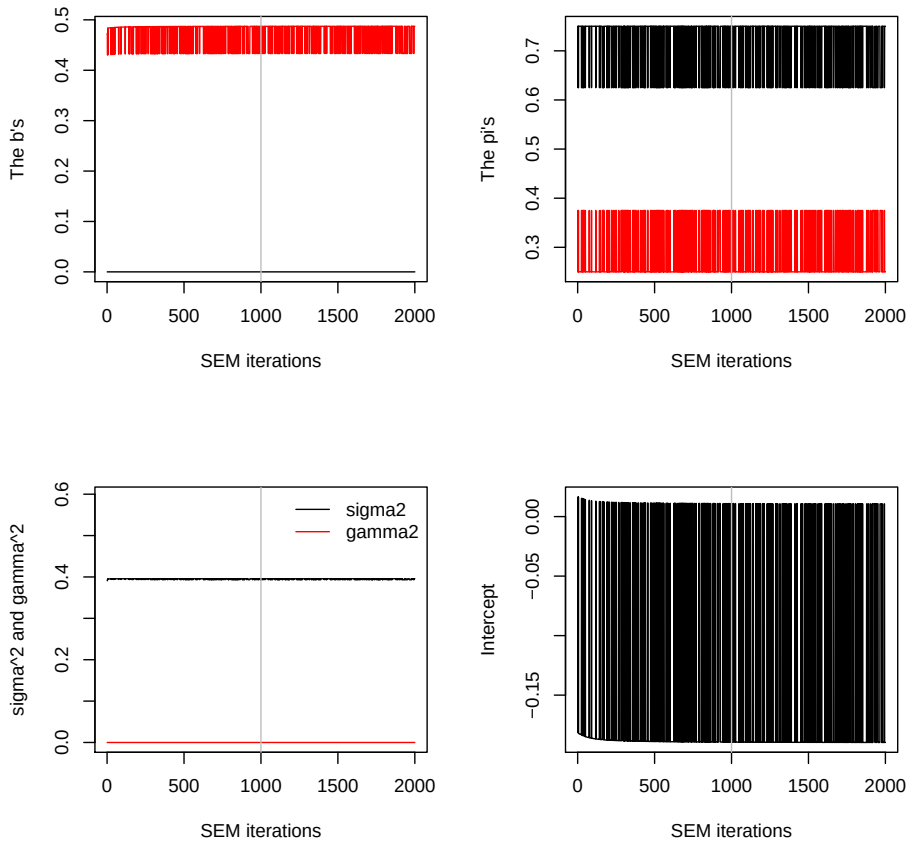


Figure 1: Values of the model parameters in view of SEM algorithm iterations. The vertical gray line in each of the four plots represents the number `nBurn` of iterations discarded before calculating maximum likelihood estimates.

(if compatible with the user's computer) and `sparse` allows to impose that one of the regression parameters is equal to 0 and thus to introduce a cluster of not significant explanatory variables.

Methods `summary()`, `plot()`, `clusters()` and `predict()`

The `summary()` method for an object returned by `fitClere()` prints an overview of the estimated parameters and returns the estimated likelihood and information based model selection criteria (AIC, BIC and ICL). The corresponding `plot()` method produces graphs such as ones presented in Figure 1.

The `clusters()` method takes one argument of class "Clere" as returned by `fitClere()` and a `threshold` argument. This function assigns each variable to the group where associated conditional probability of membership is larger than the given `threshold`. If conditional probabilities of membership are larger than the specified threshold for more than one group, then the group having the largest probability is returned and a warning is printed. If, moreover, there are several ex aequo on that largest probability, then the group with the smallest index is returned. When `threshold = NULL`, the maximum a posteriori (MAP) strategy is used to infer the clusters.

The `predict()` method has two arguments: a "Clere" object and a design matrix $\boldsymbol{X}_{new}$. Using that new design matrix, the `predict()` method returns an approximation of $\mathbb{E}\left[\boldsymbol{X}_{new}\boldsymbol{\beta}|\boldsymbol{y},\boldsymbol{X};\hat{\boldsymbol{\theta}}\right]$.

Numerical experiments

This section presents two sets of numerical experiments. The first set of experiments aims at comparing the MCEM and SEM algorithms in terms of computational time and estimation or prediction accuracy. The second set of experiments is aimed at comparing CLERE to standard dimension reduction techniques. The latter comparison is performed on both simulated and real data.

SEM algorithm versus MCEM algorithm

Description of the simulation study

In this section, a comparison between the SEM algorithm and the MCEM algorithm is performed. This comparison is performed using the four following performance indicators:

1. Computational time (CT) to run a pre-defined number of SEM/MCEM iterations. This number was set to 2,000 in this simulation study.
2. Mean squared estimation error (MSEE) defined as

$$\text{MSEE}_a = \mathbb{E}\left[(\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}_a)'(\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}_a)\right],$$

   where $a \in \{\texttt{"SEM"}, \texttt{"MCEM"}\}$ and $\widehat{\boldsymbol{\theta}}_a$ is an estimated value for parameter $\boldsymbol{\theta}$ obtained with algorithm a. Since $\boldsymbol{\theta}$ is only known up to a permutation of the group labels, we chose the permutation leading to the smallest MSEE value.
3. Mean squared prediction error (MSPE) defined as

$$\text{MSPE}_a = \mathbb{E}\left[(\boldsymbol{y}^v - \boldsymbol{X}^v\widehat{\boldsymbol{\theta}}_a)'(\boldsymbol{y}^v - \boldsymbol{X}^v\widehat{\boldsymbol{\theta}}_a)\right],$$

   where $\boldsymbol{y}^v$ and $\boldsymbol{X}^v$ are respectively a vector of responses and a design matrix from a validation data set.
4. Maximum log-likelihood (ML) reached. This quantity was approximated using 1,000 samples from $p(\boldsymbol{Z}|\boldsymbol{y};\widehat{\boldsymbol{\theta}})$.

Three versions of the MCEM algorithm were proposed for comparison with the SEM algorithm, depending on the number M (or `nsamp`) of Gibbs iterations used to approximate the *E step*. That number was varied between 5, 25 and 125. We chose these iterations numbers so as to cover different situations, from a situation in which the number of iterations is too small to a situation in which that number seems sufficient to expect having reached convergence of the simulated Markov chain. Those versions were respectively denoted MCEM_5, MCEM_{25} and MCEM_{125}. The comparison was performed using 200 simulated data sets. In order to consider high-dimensional situations with sizes allowing to reproduce multiple simulations in a reasonable time, each training data set consisted of $n = 25$ individuals and $p = 50$ variables. Validation data sets used to calculate MSPE consisted of 1,000 individuals each. All covariates were simulated independently according to the standard Gaussian distribution:

$$\forall (i,j)\ x_{ij} \sim \mathcal{N}(0,1).$$

Both training and validation data sets were simulated according to model (1) using $\beta_0 = 0$, $\boldsymbol{b} = (0,3,15)'$, $\boldsymbol{\pi} = (0.64, 0.20, 0.16)'$, $\sigma^2 = 1$ and $\gamma^2 = 0$. This is equivalent to simulate data according to the standard linear regression model defined by:

$$y_i \sim \mathcal{N}\left(\sum_{j=1}^{32} 0 \times x_{ij} + \sum_{j=33}^{42} 3 \times x_{ij} + \sum_{j=43}^{50} 15 \times x_{ij}, 1\right).$$

All algorithms were run using 10 different random starting points. Estimates yielding the largest likelihood were then used for the comparison.

Results of the comparison

Table 1 summarizes the results of the comparison between the algorithms. The MCEM_5 algorithm was 1.3 times faster than the SEM algorithm however the latter algorithm poorly performed regarding all other performance criteria (estimation quality, prediction error, likelihood maximization). This observation illustrates the importance of setting a large number M of draws to improve the estimation. Indeed, when increasing this number to 25 or 125, we observed an improvement in the estimation accuracy but no noticeable improvement in the likelihood. In turn, the SEM algorithm was quite efficient compared to the MCEM_{25} and MCEM_{125} algorithms. This algorithm not only ran faster (between 3 and 13-fold faster than MCEM_{25} and MCEM_{125} – see Table 1), but also reached predictive performances close to the oracle (i.e., using the true parameter). These good performances are mainly explained by the fact that the SEM algorithm most of the time (66.5% of the time) reached a better likelihood than the other algorithms.

The results of this simulation study were made available as an internal data set named `algoComp` in the R package **clere**. More details can be obtained using the command `help("algoComp")`.

Performance indicators	Algorithms	% of times the algorithm was best	Median (Std. Err.)
CT (seconds)	SEM	0	2.5 (0.053)
	$MCEM_5$	100	1.9 (0.016)
	$MCEM_{25}$	0	7.1 (0.027)
	$MCEM_{125}$	0	32.8 (0.121)
MSEE	SEM	58	0.31 (10.4)
	$MCEM_5$	12	20.14 (2843.3)
	$MCEM_{25}$	16.5	8.86 (3107.5)
	$MCEM_{125}$	13.5	4.02 (5664.2)
MSPE	SEM	51.5	1.3 (46.1)
	$MCEM_5$	12	75.7 (64.3)
	$MCEM_{25}$	15.5	58.7 (55.2)
	$MCEM_{125}$	21	51.6 (51.1)
	True parameter	—	1.1 (0.013)
ML	SEM	66.5	−79.3 (1.2)
	$MCEM_5$	11.5	−110.7 (2.0)
	$MCEM_{25}$	14.5	−111.6 (1.9)
	$MCEM_{125}$	7.5	−116.2 (1.7)
	True parameter	—	−77.6 (0.34)

Table 1: Performance indicators used to compare SEM and MCEM algorithms. Computational Time (CT) was measured on a Intel(R) Xeon(R) CPU E7- 4870 @ 2.40GHz processor. The best algorithm is defined as the one that either reached the largest log-likelihood (ML) or the lowest CT, Mean Squared Prediction Error (MSPE) and Mean Squared Estimation Error (MSEE).

Comparison with other methods

Description of the methods

In this section, we compare our model to standard dimension reduction approaches in terms of MSPE. Although a more detailed comparison was suggested in Yengo et al. (2014), we propose here a quick illustration of the relative predictive performance of our model. The comparison is achieved using data simulated according to the scenario described above in Section SEM algorithm versus MCEM algorithm. The methods selected for comparison are the Ridge regression (Hoerl and Kennard, 1970), the Elastic net (Zou and Hastie, 2005), the LASSO (Tibshirani, 1996), PACS (Sharma et al., 2013), the method of Park and colleagues (Park et al., 2007, subsequently denoted AVG) and the Spike and Slab model (Ishwaran and Rao, 2005, subsequently denoted SS). The first three methods are implemented in the freely available R package **glmnet**. With the latter package, the tuning parameter `lambda` was selected using the function `cv.glm` (with 5 folds) aiming at minimizing the mean squared error (option `type = "mse"`). In particular for the Elastic net, the second tuning parameter `alpha` (measuring the amount of mixture between the L^1 and L^2 penalty) was jointly selected with `lambda` to minimize the mean squared error. The R package **glmnet** proposes a procedure for automatically selecting values for `lambda`. We therefore used this default procedure while we selected `alpha` among $\{0, 0.1, 0.2, \ldots, 0.9, 1\}$. The PACS methodology proposes to estimate the regression coefficients by solving a penalized least squares problem. It imposes a constraint on $\boldsymbol{\beta}$ that is a weighted combination of the L^1 norm and the pairwise L^∞ norm. Upper-bounding the pairwise L^∞ norm enforces the covariates to have close coefficients. When the constraint is strong enough, closeness translates into equality achieving thus a grouping property. For PACS, no software was available. Only an R script was released on Bondell's web page[1]. Since this R script was running very slowly, we decided to reimplement it in C++ and observed a 30-fold speed-up of computational time. Similarly to Bondell's R script, our implementation uses two parameters `lambda` and `betawt`. Our reimplementation of Bondell's script was included in the R package **clere** in the function `fitPacs()`. In Sharma et al. (2013), the authors suggest assigning `betawt` with the coefficients obtained from a ridge regression model

[1] http://www4.stat.ncsu.edu/~bondell/Software/PACS/PACS.R.r

after the tuning parameter was selected using AIC. In this simulation study we used the same strategy; however the ridge parameter was selected via 5-fold cross validation. 5-fold CV was preferred to AIC since selecting the ridge parameter using AIC always led to estimated coefficients equal to zero. Once `betawt` was selected, `lambda` was chosen via 5-fold cross validation among the following values: 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100, 200 and 500. All other default parameters of their script were unchanged. The AVG method is a two-step approach. The first step uses hierarchical clustering of covariates to create surrogate covariates by averaging the variables within each group. Those new predictors are afterwards included in a linear regression model, replacing the primary variables. A variable selection algorithm is then applied to select the most predictive groups of covariates. To implement this method, we followed the algorithm described in Park et al. (2007) and programmed it in R. The Spike and Slab model is a Bayesian approach for variable selection. It is based on the assumption that the regression coefficients are distributed according to a mixture of two centered Gaussian distributions with different variances. One component of the mixture (the spike) is chosen to have a small variance, while the other component (the slab) is allowed to have a large variance. Variables assigned to the spike are dropped from the model. We used the R package **spikeslab** to run the Spike and Slab models. Especially, we used the function **spikeslab** from that package to detect influential variables. The number of iterations used to run the function **spikeslab** was 2,000 (1,000 discarded).

When running `fitClere()`, the number `nItEM` of SEM iterations was set to 2,000. The number `g` of groups for CLERE was chosen between 1 and 5 using AIC (option `analysis = "aic"`). Two versions of CLERE were considered: the one with all parameters estimated and the one with b_1 set to 0. The latter approach is subsequently denoted CLERE_0 (option `sparse = TRUE`).

Results of the comparison

Figure 2 summarizes the comparison between the methods. In this simulated scenario, CLERE outperformed the other methods in terms of prediction error. These good performances were improved when parameter b_1 was set to 0. CLERE was also the most parsimonious approach with an averaged number of estimated parameters equal to 7.7 (6.9 when $b_1 = 0$). The second best approach was PACS which also led to parsimonious models. The superiority of such methods could be expected since in the simulation model the regression coefficients are gathered in three groups. Overall variable selection approaches yielded the largest prediction error in this simulation. CLERE, PACS and Spike and Slab had the largest computational times (CT). For CLERE and PACS this loss in CT was compensated by a a strong improvement in prediction error as explained above, while Spike and Slab yielded the worst prediction error in addition to being the slowest approach in this example.

The results of this simulation study were made available as an internal data set in the R package **clere**. The object is called `numExpSimData` and more details can be obtained using the command `help("numExpSimData")`.

Real data sets analysis

Description of the data sets

We used in this section the real data sets `Prostate` and `eyedata` from the R packages **lasso2** (Lokhorst et al., 2014) and **flare** (Li et al., 2014) respectively. The `Prostate` data set comes from a study that examined the correlation between the level of prostate specific antigen and a number of clinical measures in $n = 97$ men who were about to receive a radical prostatectomy. This data set is a benchmark data set used in multiple publications about high-dimensional regression model, including Tibshirani (1996); Hastie et al. (2001), and was chosen here in order to illustrate the performance of CLERE in comparison to the competing methods. We used the prostate specific antigen (variable `lpsa`) as response variable and the $p = 8$ other measurements as covariates.

The `eyedata` data set is extracted from the published study of Scheetz et al. (2006). This data set consists of gene expression levels measured at $p = 200$ probes in $n = 120$ rats. The response variable utilized was the expression of the TRIM32 gene which is a biomarker of the Bardet-Bidel Syndrome (BBS). We chose this data set to illustrate the performances of CLERE on a (manageable) high-dimensional problem which is the actual context for which this method was developped (Yengo et al., 2014).

Those two data sets were utilized to compare CLERE to the same methods used in the previous section where the simulation study was presented. All methods were compared in terms of out-of-sample prediction error estimated using 5-fold cross validation (CV). Those CV statistics were then averaged and compared across the methods in Table 2.

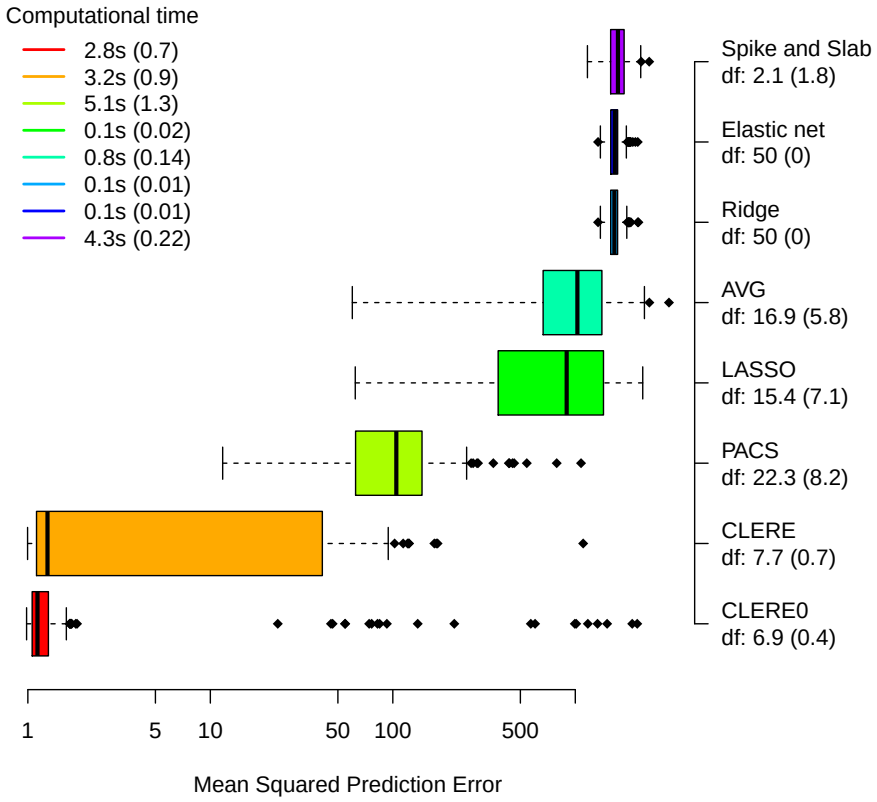


Figure 2: Comparison between CLERE and some standard dimension reduction approaches. The number of estimated parameters (df: $+/-$ standard error) is given in the right along with the name of the method utilized. The average computational time with its corresponding standard error (given in parenthesis) is also provided for each situation.

Running the analysis

Before presenting the results of the comparison between CLERE and its competitors, we illustrate the commands to run the analysis of the `Prostate` data set. The data set is loaded by typing:

```
R> data("Prostate", package = "lasso2")
R> y <- Prostate[, "lpsa"]
R> x <- as.matrix(Prostate[, -which(colnames(Prostate) == "lpsa")])
```

Possible training (`xt` and `yt`) and validation (`xv` and `yv`) sets are generated as follows:

```
R> itraining <- 1:(0.8*nrow(x))
R> xt <- x[ itraining,]; yt <- y[ itraining]
R> xv <- x[-itraining,]; yv <- y[-itraining]
```

The `fitClere()` function is run using the AIC to select the number of groups between 1 and 5. To lessen the impact of local minima in the model selection, 5 random starting points are used. This can be implemented by:

```
R> Seed <- 1234
R> mod  <- fitClere(y = yt, x = xt, g = 5, analysis = "aic", parallel = TRUE,
+                   nstart = 5, sparse = TRUE, nItEM = 2000, nBurn = 1000,
+                   nItMC = 10, dp = 5, nsamp = 1000, seed = Seed)
R> summary(mod)
   -------------------------------
   | CLERE | Yengo et al. (2013) |
   -------------------------------

   Model object 2 groups of variables ( Selected using AIC criterion )
   ---
    Estimated parameters using SEM algorithm are
```

```
 intercept = -0.1339
 b         = 0.0000   0.4722
 pi        = 0.7153   0.2848
 sigma2    = 0.395
 gamma2    = 4.065e-08

---
 Log-likelihood =  -78.31
 Entropy        =  0.5464
 AIC            =  168.63
 BIC            =  182.69
 ICL            =  183.23
```

```
R> plot(mod)
```

Running the command `plot(mod)` generates the plot given in Figure 1. We can also access the cluster memberships by running the command `clusters()`. For example, running the command `clusters(mod,threshold = 0.7)` yields

```
R> clusters(mod, thresold = 0.7)
lcavol lweight     age    lbph     svi     lcp gleason   pgg45
     2       2       1       1       1       1       1       1
```

In the example above 2 variables, being the cancer volume (`lcavol`) and the prostate weight (`lweight`), were assigned to group 2 ($b_2 = 0.4737$). The other 6 variables were assigned to group 1 ($b_1 = 0$). Posterior probabilities of membership are available through the slot `P` in the object of class "Clere".

```
R> mod@P
        Group 1 Group 2
lcavol    0.000   1.000
lweight   0.000   1.000
age       1.000   0.000
lbph      1.000   0.000
svi       0.764   0.236
lcp       1.000   0.000
gleason   1.000   0.000
pgg45     1.000   0.000
```

The covariates were respectively assigned to their group with a probability larger than 0.7. Moreover, given that parameter γ^2 had a very small value ($\widehat{\gamma^2} = 4.065 \times 10^{-8}$), we can argue that cancer volume and prostate weight are the only relevant explanatory covariates. To assess the prediction error associated with the model we can run the command `predict()` as follows:

```
R> error <- mean((yv - predict(mod, xv))^2)
R> error
[1] 1.543122
```

Results of the analysis

Table 2 summarizes the prediction errors and the number of parameters obtained for all the methods. CLERE_0 had the lowest prediction error in the analysis of the `Prostate` data set and the second best performance for the `eyedata` data set. The AVG method was also very competitive compared to the variable selection approaches stressing thus the relevance of creating groups of variables to reduce the dimensionality (especially in the `eyedata` data set). It is worth noting that in both data sets, imposing the constraint $b_1 = 0$ improved the predictive performance of CLERE.

In the `Prostate` data set, CLERE robustly identified two groups of variables representing influential ($b_2 > 0$) and not relevant variables ($b_1 = 0$). In the `eyedata` data set in turn, AIC led to selecting only one group of variables. However, this did not lessen the predictive performance of the model since CLERE_0 was second best after AVG, while needing significantly less parameters. PACS performed badly in both data sets. The Elastic net showed good predictive performances compared to the variable selection methods like LASSO or the Spike and Slab model. Ridge regression and Elastic net had comparable results in both data sets. In line with the results of the simulation study, we observed that despite a larger computational time (CT), CLERE and CLERE_0 had a reduced mean squared error compared to the fastest methods. However, this improvement was less substantial than observed in the simulation study given the differences in CT. This increased CT may be explained by the fact that no simple stopping rule is proposed when fitting CLERE. We may therefore contemplate that a smaller

	100×Averaged CV statistic (Std. Error)	Number of parameters (Std. Error)	CT (seconds) (Std. Error)
`Prostate data set`			
LASSO	90.2 (29)	5.6 (0.7)	0.064 (0.007)
RIDGE	86.8 (24)	8.0 (0)	0.065 (0.002)
Elastic net	90.3 (24)	8.0 (0)	0.065 (0.002)
STEP	442.4 (137)	8.0 (0)	0.004 (0.001)
CLERE	82.4 (25)	6.0 (0)	1.1 (0.1)
CLERE_0	74.5 (26)	5.0 (0)	2.7 (0.8)
Spike and Slab	85.6 (26)	5.6 (0.7)	4.2 (0.03)
AVG	90.2 (27)	6.2 (0.4)	0.44 (0.06)
PACS	90.6 (34)	5.6 (0.4)	0.053 (0.002)
`eyedata`			
LASSO	0.73 (0.1)	21.2 (2)	0.18 (0.01)
RIDGE	0.74 (0.1)	200.0 (0)	0.24 (0.004)
Elastic net	0.74 (0.1)	200.0 (0)	0.23 (0.003)
STEP	1142.6 (736)	95.0 (0)	0.083 (0.002)
CLERE	0.73 (0.1)	4.0 (0)	21.5 (0.2)
CLERE_0	0.72 (0.1)	3.0 (0)	21.1 (0.1)
Spike and Slab	0.81 (0.2)	12.4 (0.9)	10.3 (0.1)
AVG	0.70 (0.04)	15.6 (2)	10.6 (0.4)
PACS	2.0 (0.9)	3.0 (0.3)	108.9 (28)

Table 2: Real data analysis. Out-of-sample prediction error (averaged CV statistic) was estimated using 100-folds cross validation. The number of parameters reported for CLERE/CLERE_0 was selected using AIC. CT stands for the average Computational Time.

number of SEM iterations could have been used to yield a similar prediction error. Indeed, when looking at Figure 1, we see that convergence was achieved almost from the first 10 iterations. Still, the observed CT for CLERE being around 22s for the `eyedata` data set and around 3s for the `Prostate` data set remains affordable in these examples.

The results of this analysis on real data were made available as an internal data set named `numExpRealData` in the R package **clere**. Using the command `help("numExpRealData")` more details can be obtained.

Conclusions

We presented in this paper the R package **clere**. This package implements two efficient algorithms for fitting the CLusterwise Effect REgression model: the MCEM and the SEM algorithms. The MCEM algorithm is to be preferred when $p < n$; the SEM algorithm is more efficient for high-dimensional data sets ($n < p$). The good performance of SEM over MCEM could have been expected regarding the computational complexities of the two algorithms that are $\mathcal{O}\left(npg + g^3 + N_{max}ng\right)$ and $\mathcal{O}\left(M(p^2 + pg)\right)$ respectively. In fact, as long as $p > n$, the SEM algorithm has a lower complexity. However, the computational time to run our SEM algorithm is more variable compared to MCEM as its M step does not have a closed form. We finally advocate the use of the MCEM algorithm only when $p \ll n$. Another important feature for model interpretation is proposed by constraining the model parameter b_1 to equal 0, which leads to variable selection. Such a constraint may also lead to a reduced prediction error. We illustrated on a real data set, how to run an analysis, based on a detailed R script. Although our numerical experiments showed that the CLERE method tended to be slower than variable selection methods, it still provided better or competitive predictive performance. In addition, the CLERE model was often more parsimonious than other models and was easily interpretable since groups of regression coefficients/variables could be summarized using a single parameter.

Our model can easily be extended to the analysis of binary responses. This extension will be made available in a forthcoming version of the package. Another direction for future research will be to develop an efficient stopping rule for the proposed SEM algorithm, specific to our context. Such a criterion is expected to improve the computational performance of our estimation algorithm.

Bibliography

H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974. [p97]

D. Bates and D. Eddelbuettel. Fast and elegant numerical linear algebra using the RcppEigen package. *Journal of Statistical Software*, 52(5):1–24, 2013. URL http://www.jstatsoft.org/v52/i05. [p92]

C. Biernacki and J. Jacques. A generative model for rank data based on insertion sort algorithm. *Computational Statistics and Data Analysis*, 58:162–176, 2013. [p96]

C. Biernacki, G. Celeux, and G. Goavert. Assessing a mixture model for clustering with the integrated completed likelihood. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(7):719–725, 2000. [p97]

H. D. Bondell and B. J. Reich. Simultaneous regression shrinkage, variable selection, and supervised clustering of predictors with OSCAR. *Biometrics*, 64:115–123, 2008. [p92]

G. Casella. An introduction to empirical Bayes data analysis. *The American Statistician*, 39(2):83–87, 1985. [p93]

G. Celeux, D. Chauveau, and J. Diebolt. Some stochastic versions of the EM algorithm. *Journal of Statistical Computation and Simulation*, 55:287–314, 1996. [p93, 95]

A. P. Dempster, M. N. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39:1–22, 1977. [p93]

D. Eddelbuettel and R. François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL http://www.jstatsoft.org/v40/i08/. [p92]

J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010. URL http://www.jstatsoft.org/v33/i01/. [p92]

G. Govaert and M. Nadif. Block clustering with Bernoulli mixture models: Comparison of different approaches. *Computational Statistics and Data Analysis*, 52:3233–3245, 2008. [p93]

A. Gunawardana and W. Byrne. Convergence theorems for generalized alternating minimization procedures. *Journal of Machine Learning Research*, 6:2049–2073, 2005. [p93]

T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, New York, 2001. [p101]

A. E. Hoerl and W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12:55–67, 1970. [p92, 100]

H. Ishwaran and J. S. Rao. Spike and slab variable selection: Frequentist and Bayesian strategies. *The Annals of Statistics*, 33(2):730–773, 2005. [p92, 100]

H. Ishwaran, J. S. Rao, and U. B. Kogalur. *spikeslab: Prediction and Variable Selection Using Spike and Slab Regression*, 2013. URL https://CRAN.R-project.org/package=spikeslab. R package version 1.1.5. [p92]

I. T. Jolliffe. A note on the use of principal components in regression. *Journal of the Royal Statistical Society C*, 31(3):300–303, 1982. [p92]

R. A. Levine and G. Casella. Implementations of the Monte Carlo EM algorithm. *Journal of Computational and Graphical Statistics*, 10(3):422–439, 2001. [p93]

X. Li, T. Zhao, L. Wang, X. Yuan, and H. Liu. *flare: Family of Lasso Regression*, 2014. URL https://CRAN.R-project.org/package=flare. R package version 1.5.0. [p101]

J. Lokhorst, B. Venables, and B. Turlach. *lasso2: L1 Constrained Estimation aka 'LASSO'*, 2014. URL https://CRAN.R-project.org/package=lasso2. R package version 1.2-19; port to R and tests, etc.: Martin Maechler. [p101]

M. Mariadassou, S. Robin, and C. Vacher. Uncovering latent structure in valued graphs: A variational approach. *The Annals of Applied Statistics*, 4(2):715–742, 2010. [p93]

M. Y. Park, T. Hastie, and R. Tibshirani. Averaged gene expressions for regression. *Biostatistics*, 8: 212–227, 2007. [p100, 101]

T. E. Scheetz, K.-Y. A. Kim, R. E. Swiderski, A. R. Philp, T. A. Braun, K. L. Knudtson, A. M. Dorrance, G. F. DiBona, J. Huang, T. L. Casavant, V. C. Sheffield, and E. M. Stone. Regulation of gene expression in the mammalian eye and its relevance to eye disease. *Proceedings of the National Academy of Sciences of the United States of America*, 103(39):14429–14434, 2006. [p101]

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6:461–464, 1978. [p97]

S. Searle, G. Casella, and C. McCulloch. *Variance Components*. Wiley Series in Probability and Mathematical Statistics: Applied Probability and Statistics. Wiley, 1992. [p96]

D. B. Sharma, H. D. Bondell, and H. H. Zhang. Consistent group identification and variable selection in regression with correlated predictors. *Journal of Computational and Graphical Statistics.*, 22(2):319–340, 2013. [p92, 100]

The MathWorks Inc. *MATLAB – The Language of Technical Computing, Version R2014b.* Natick, Massachusetts, 2014. URL http://www.mathworks.com/products/matlab/. [p92]

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society B*, 58:267–288, 1996. [p92, 100, 101]

C. G. Wei and M. Tanner. A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms. *Journal of the American Statistical Association*, 85:699–704, 1990. [p93]

L. Yengo and M. Canouil. *clere: Simultaneous Variables Clustering and Regression*, 2015. URL https://CRAN.R-project.org/package=clere. R package version 1.1.4. [p92]

L. Yengo, J. Jacques, and C. Biernacki. Variable clustering in high dimensional linear regression models. *Journal de la Société Française de Statistique*, 155(2):38–56, 2014. [p92, 100, 101]

H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society B*, 67:301–320, 2005. [p92, 100]

Loïc Yengo
Integrated Genomics and Metabolic Diseases Modeling
CNRS UMR 8199 - Lille Institute of Biology
E.G.I.D – FR3508 European Genomics Institute of Diabetes
1, rue du Professeur Calmette, BP 447, 59021 Lille cedex
France
loic.yengo@cnrs.fr

Julien Jacques
ERIC Laboratory
Université de Lyon - Lumière
5 avenue Pierre Mendès France, 69676 Bron cedex
France
julien.jacques@univ-lyon2.fr

Christophe Biernacki
MODAL team (Inria) & Laboratoire Paul Painlevé (UMR CNRS 8524)
University Lille I
Cité Scientifique, 59655 Villeneuve d'Ascq cedex
France
christophe.biernacki@math.univ-lille1.fr

Mickael Canouil
Integrated Genomics and Metabolic Diseases Modeling
CNRS UMR 8199 – Lille Institute of Biology
E.G.I.D – FR3508 European Genomics Institute of Diabetes
1, rue du Professeur Calmette, BP 447, 59021 Lille cedex
France
mickael.canouil@cnrs.fr

Stylometry with R: A Package for Computational Text Analysis

by Maciej Eder, Jan Rybicki and Mike Kestemont

Abstract This software paper describes 'Stylometry with R' (**stylo**), a flexible R package for the high-level analysis of writing style in stylometry. Stylometry (computational stylistics) is concerned with the quantitative study of writing style, e.g. authorship verification, an application which has considerable potential in forensic contexts, as well as historical research. In this paper we introduce the possibilities of **stylo** for computational text analysis, via a number of dummy case studies from English and French literature. We demonstrate how the package is particularly useful in the exploratory statistical analysis of texts, e.g. with respect to authorial writing style. Because **stylo** provides an attractive graphical user interface for high-level exploratory analyses, it is especially suited for an audience of novices, without programming skills (e.g. from the Digital Humanities). More experienced users can benefit from our implementation of a series of standard pipelines for text processing, as well as a number of similarity metrics.

Introduction

Authorship is a topic which continues to attract considerable attention with the larger public. This claim is well illustrated by a number of high-profile case studies that have recently made headlines across the popular media, such as the attribution of a pseudonymously published work to acclaimed *Harry Potter* novelist, J. K. Rowling (Juola, 2013), or the debate surrounding the publication of Harper Lee's original version of *To Kill a Mocking Bird* and the dominant role which her editor might have played therein (Gamerman, 2015). The authorship of texts clearly matters to readers across the globe (Love, 2002) and therefore it does not come as a surprise that computational authorship attribution increasingly attracts attention in science, because of its valuable real-world applications, for instance, related to forensics topics such as plagiarism detection, unmasking the author of harassment messages or even determining the provenance of bomb letters in counter-terrorism research. Interestingly, the methods of stylometry are also actively applied in the Humanities, where multiple historic authorship problems in literary studies still seek a definitive solution – the notorious Shakespeare-Marlowe controversy is perhaps the best example in this respect.

Authorship attribution plays a prominent role in the nascent field of stylometry, or the computational analysis of writing style (Juola, 2006; Stamatatos et al., 2000; Stamatatos, 2009; Koppel et al., 2009; Van Halteren et al., 2005). While this field has important historical precursors (Holmes, 1994, 1998), recent decades have witnessed a clear increase in the scientific attention for this problem. Because of its emergent nature, replicability and benchmarking still pose significant challenges in the field (Stamatatos, 2009). Publicly available benchmark data sets are hard to come across, mainly because of copyright and privacy issues, and there are only a few stable, cross-platform software packages out there which are widely used in the community. Fortunately, a number of recent initiatives lead the way in this respect, such as the recent authorship tracks in the PAN competition (`http://pan.webis.de`), where e.g. relevant data sets are efficiently interchanged.

In this paper we introduce 'Stylometry with R' (**stylo**), a flexible R package for the high-level stylistic analysis of text collections. This package explicitly seeks to further contribute to the recent development in the field towards a more advanced level of replicability and benchmarking in the field. Stylometry is a multidisciplinary research endeavor, attracting contributions from divergent scientific domains, which include researchers from Computer Science – with a fairly technical background – as well as experts from the Humanities – who might lack the computational skills which would allow them easy access to the state-of-the-art methods in the field (Schreibman et al., 2004). Importantly, this package has the potential to help bridge the methodological gap luring between these two communities of practice: on the one hand, **stylo**'s API allows to set up a complete processing pipeline using traditional R scripting; on the other hand, **stylo** also offers a rich graphical user interface which allows non-technical, even novice practitioners to interface with state-of-the-art methods without the need for any programming experience.

Overview of stylometry

Stylometry deals with the relationship between the writing style in texts and meta-data about those texts (such as date, genre, gender, authorship). Researchers in 'stylochronometry', for instance, are interested in inferring the date of composition of texts on the basis of stylistic aspects (Stamou, 2008;

Juola, 2007). Authorship studies are currently the most popular application of stylometry. From the point of view of literary studies, stylometry is typically concerned with a number of recent techniques from computational text analysis that are sometimes termed 'distant reading', 'not reading' or 'macroanalysis' (Jockers, 2013). Instead of the traditional practice of 'close reading' in literary analysis, stylometry does not set out from a single direct reading; instead, it attempts to explore large text collections using computational techniques (and often visualization). Thus, stylometry tries to expand the scope of inquiry in the humanities by scaling up research resources to large text collections in order to find relationships and patterns of similarity and difference invisible to the eye of the human reader.

Usually, stylometric analyses involve a complex, multi-stage pipeline of (i) preprocessing, (ii) feature extraction, (iii) statistical analysis, and finally, (iv) presentation of results, e.g. via visualization. To this end, researchers presently have to resort to an ad hoc combination of proprietary, language-dependent tools that cannot easily be ported across different platforms. Such solutions are difficult to maintain and exchange across (groups of) individual researchers, preventing straightforward replication of research results and reuse of existing code. **stylo**, the package presented, offers a rich, user-friendly suite of functionality that is ideally suited for fast exploratory analysis of textual corpora as well as classification tasks such as are needed in authorship attribution. The package offers an implementation of the main methods currently dominant in the field. Its main advantage therefore lies in the integration of typical (e.g. preprocessing) procedures from stylometry and statistical functionality by other, external libraries. Written in the R language, the source code and binaries for the package are freely available from the Comprehensive R Archive Network, guaranteeing a straightforward installation process across different platforms (both Unix- and Windows-based operating systems). The code is easily adaptable and extensible: the developers therefore continue to welcome user contributions, feedback and feature requests. Our code is open source and GPL-licensed: it is being actively developed on GitHub.[1]

In the rest of this paper, we will first illustrate the functionality of the package for unsupervised multivariate analysis through the high-level function `stylo()`. Secondly, we will discuss a number of graphical user interfaces which we provide for quick exploration of corpora, in particular by novice users or students in an educational setting, as well as for scholars in the Humanities without programming experience. Next, we move on to the function `classify()`, implementing a number of supervised classification procedures from the field of Machine Learning. Finally, we concisely discuss the `oppose()`, `rolling.delta()` and `rolling.classify()` functionality which allow, respectively, to inspect differences in word usage between two subsets of a corpus, and to study the evolution of the writing style in a text.

Overview of the package

Downloading, installing and loading **stylo** is straightforward. The package is available at CRAN and at GitHub repository. The main advantages and innovative features of **stylo** include:

Feature extraction

Crucial in stylometry is the extraction of quantifiable features related to the writing style of texts (Sebastiani, 2002). A wide range of features have been proposed in the literature, considerably varying in complexity (Stamatatos, 2009). 'Stylometry with R' focuses on features that can be automatically extracted from texts, i.e. without having to resort to language-dependent preprocessing tools. The features that the package allows to extract are *n*-grams on token- and character level (Houvardas and Stamatatos, 2006; Kjell, 1994). Apart from the fact that this makes the package considerably language-independent, such shallow features have been shown to work well for a variety of tasks in stylometry (Daelemans, 2013; Kestemont, 2014). Moreover, users need not annotate their text materials using domain-specific tools before analyzing them with 'Stylometry with R'. Apart from the standard usage, however, the package does allow the users to load their own annotated corpora, provided that this is preceded by some text pre-processing tasks. An example of such a non-standard procedure will be shown below. Thus, **stylo** does not aim to supplant existing, more targeted tools and packages from Natural Language Processing (Feinerer et al., 2008) but it can easily accommodate the output of such tools as a part of its processing pipeline.

[1] https://github.com/computationalstylistics/stylo

Metrics

A unique feature of **stylo** is that it offers reference implementations for a number of established distance metrics from multivariate statistical analysis, which are popular in stylometry, but uncommon outside the field. Burrows's Delta is the best example here (Burrows, 2002); it is an intuitive distance metric which has attracted a good share of attention in the community, also from a theoretical point of view (Hoover, 2004a,b; Argamon, 2011).

Graphical user interface

The high-level functions of the package provide a number of Graphical User Interfaces (GUIs) which can be used to intuitively set up a number of established experimental workflows with a few clicks (e.g. unsupervised visualization of texts based on word frequencies). These interfaces can be easily invoked from the command line in R and provide an attractive overview of the various experimental parameters available, allowing users to quickly explore the main stylistic structure of corpora. This feature is especially useful in an educational setting, allowing (e.g. undergraduate) students from different fields, typically without any programming experience, to engage in stylometric experimentation. The said high-level functions keep the analytic procedure from corpus pre-processing to final results presentation manageable from within a single GUI. More flexibility, however, can be achieved when the workflow is split into particular steps, each controlled by a dedicated lower-level function from the package, as will be showcased below.

Example workflow

An experiment in stylometry usually involves a workflow whereby, subsequently, (i) textual data is acquired, (ii) the texts are preprocessed, (iii) stylistic features are extracted, (iv) a statistical analysis is performed, and finally, (v) the results are outputted (e.g. visualized). We will now illustrate how such a workflow can be performed using the package.

Corpus preparation

One of the most important features of **stylo** is that it allows loading textual data either from R objects, or directly from corpus files stored in a dedicated folder. Metadata of the input texts are expected to be included in the file names. The file name convention assumes that any string of characters followed by an underscore becomes a class identifier (case sensitive). In final scatterplots and dendrograms, colors of the samples are assigned according to this convention; common file extensions are dropped. E.g. to make the samples colored according to authorial classes, files might be named as follows:

```
ABronte_Agnes.txt   ABronte_Tenant.txt      Austen_Pride.txt
Austen_Sense.txt    Austen_Emma.txt         CBronte_Professor.txt
CBronte_Jane.txt    CBronte_Villette.txt    EBronte_Wuthering.txt
```

All examples below can be reproduced by the user on data sets which can be downloaded from the authors' project website.[2] For the sake of convenience, however, we will use the datasets that come with the package itself:

```
data(novels)
data(galbraith)
data(lee)
```

Our first example uses nine prose novels by Jane Austen and the Brontë sisters, provided by the dataset `novels`.

Preprocessing

stylo offers a rich set of options to load texts in various formats from a file system (preferably encoded in UTF-8 Unicode, but it also supports other encodings, e.g. under Windows). Apart from raw text, **stylo** allows to load texts encoded according to the guidelines of the Text Encoding Initiative, which is relatively prominent in the community of text analysis researchers.[3] To load all the files saved in a directory (e.g. 'corpus_files'), users can use the following command:

[2] https://sites.google.com/site/computationalstylistics/corpora
[3] http://www.tei-c.org/index.xml

```
raw.corpus <- load.corpus(files = "all", corpus.dir = "corpus_files",
         encoding = "UTF-8")
```

If the texts are annotated in e.g. XML, an additional pre-processing procedure might be needed:

```
corpus.no.markup <- delete.markup(raw.corpus, markup.type = "xml")
```

Since the dataset that we will use has no annotation, the markup deletion can be omitted. We start the procedure with making the data visible for the user:

```
data(novels)
summary(novels)
```

To preprocess the data, **stylo** offers a number of tokenizers that support a representative set of European languages, including English, Latin, German, French, Spanish, Dutch, Polish, Hungarian, as well as basic support for non-Latin alphabets such as Korean, Chinese, Japanese, Hebrew, Arabic, Coptic and Greek. Tokenization refers to the process of dividing a string of input texts into countable units, such as word tokens. To tokenize the English texts, e.g. splitting items as 'don't' into 'do' and 'n't' and lowercasing all words, the next command is available:

```
tokenized.corpus <- txt.to.words.ext(novels, language = "English.all",
            preserve.case = FALSE)
```

The famous first sentence of Jane Austen's *Pride and Prejudice*, for instance, looks like this in its tokenized version (the 8th to the 30th element of the corresponding vector):

```
tokenized.corpus$Austen_Pride[8:30]
```

```
 [1] "it"           "is"           "a"            "truth"        "universally"
 [6] "acknowledged" "that"         "a"            "single"       "man"
[11] "in"           "possession"   "of"           "a"            "good"
[16] "fortune"      "must"         "be"           "in"           "want"
[21] "of"           "a"            "wife"
```

To see basic statistics of the tokenized corpus (number of texts/samples, number of tokens in particular texts, etc.), one might type:

```
summary(tokenized.corpus)
```

For complex scripts, such as Hebrew, custom splitting rules could easily be applied:

```
tokenized.corpus.custom.split <- txt.to.words(tokenized.corpus,
       splitting.rule = "[^A-Za-z\U05C6\U05D0-\U05EA\U05F0-\U05F2]+",
       preserve.case = TRUE)
```

A next step might involve 'pronoun deletion'. Personal pronouns are often removed in stylometric studies because they tend to be too strongly correlated with the specific topic or genre of a text (Pennebaker, 2011), which is an unwanted artefact in e.g. authorship studies (Hoover, 2004a,b). Lists of pronouns are available in **stylo** for a series of languages supported. They can be accessed via for example:

```
stylo.pronouns(language = "English")
```

```
 [1] "he"         "her"        "hers"       "herself"    "him"
 [6] "himself"    "his"        "i"          "me"         "mine"
[11] "my"         "myself"     "our"        "ours"       "ourselves"
[16] "she"        "thee"       "their"      "them"       "themselves"
[21] "they"       "thou"       "thy"        "thyself"    "us"
[26] "we"         "ye"         "you"        "your"       "yours"
[31] "yourself"
```

Removing pronouns from the analyses (much like stopwords are removed in Information Retrieval analyses) is easy in **stylo**, using the `delete.stop.words()` function:

```
corpus.no.pronouns <- delete.stop.words(tokenized.corpus,
              stop.words = stylo.pronouns(language = "English"))
```

The above procedure can also be used to exclude any set of words from the input corpus.

Features

After these preprocessing steps, users will want to extract gaugeable features from the corpus. In a vast majority of approaches, stylometrists rely on high-frequency items. Such features are typically extracted in the level of (groups of) words or characters, called n-grams (Kjell, 1994). Both word-token and character n-grams are common textual features in present-day authorship studies. **Stylo** allows users to specify the size of the n-grams which they want to use. For third order character trigrams ($n = 3$), for instance, an appropriate function of **stylo** will select partially overlapping series of character groups of length 3 from a string of words (e.g. 'tri', 'rig', 'igr', 'gra', 'ram', 'ams'). Whereas token level features have a longer tradition in the field, character n-grams have been fairly recently borrowed from the field of language identification in Computer Science (Stamatatos, 2009; Eder, 2011). Both n-grams at the level of characters and words have been listed among the most effective stylistic features in survey studies in the field. For $n = 1$, such text representations model texts under the so-called 'bag-of-words' assumption that the order and position of items in a text is negligible stylistic information. To convert single words into third order character chains, or trigrams:

```
corpus.char.3.grams <- txt.to.features(corpus.no.pronouns, ngram.size = 3,
       features = "c")
```

Sampling

Users can study texts in their entirety, but also draw consecutive samples from texts in order to effectively assess the internal stylistic coherence of works. The sampling settings will affect how the relative frequencies are calculated and allow users to normalize text length in the data set. Users can specify a sampling size (expressed in current units, e.g. words) to divide texts into consecutive slices. The samples can partially overlap and they can be also be extracted randomly. As with all functions, the available options are well-documented:

```
help(make.samples)
```

To split the current corpus into non-overlapping samples of 20,000 words each, one might type:

```
sliced.corpus <- make.samples(tokenized.corpus, sampling = "normal.sampling",
        sample.size = 20000)
```

Counting frequent features

A crucial point of the dataset preparation is building a frequency table. In stylometry, analyses are typically restricted to a feature space containing the n most frequent items. It is relatively easy to extract e.g. the 3,000 most frequent features from the corpus using the following function:

```
frequent.features <- make.frequency.list(sliced.corpus, head = 3000)
```

After the relevant features have been harvested, users have to extract a vector for each text or sample, containing the relative frequencies of these features, and combine them into a frequency table for the corpus. Using an appropriate function from **stylo**, these vectors are combined in a feature frequency table which can be fed into a statistical analysis (external tables of frequencies can be loaded as well):

```
freqs <- make.table.of.frequencies(sliced.corpus, features = frequent.features)
```

Feature selection and sampling settings might interact: an attractive unique feature of **stylo** is that it allows users to specify different 'culling' settings. Via culling, users can specify the percentage of samples in which a feature should be present in the corpus in order to be included in the analysis. Words that do not occur in at least the specified proportion of the samples in the corpus will be ignored. For an 80% culling rate, for instance:

```
culled.freqs <- perform.culling(freqs, culling.level = 80)
```

Analysis

Stylo offers a seamless wrapper for a variety of established statistical routines available from R's core library or contributed by third-party developers; these include t-Distributed Stochastic Neighbor Embedding (van der Maaten and Hinton, 2008), Principal Components Analysis, Hierarchical Clustering and Bootstrap Consensus Trees (a method which will be discussed below). An experiment can be initiated with a pre-existing frequency table with the following command:

```
stylo(frequencies = culled.freqs, gui = FALSE)
```

When the input documents are loaded directly from text files, the default features are most frequent words (MFWs), i.e. 1-grams of frequent word forms turned into lowercase. Also, by default, a standard cluster analysis of the 100 most frequent features will be performed. To perform e.g. a Principal Components Analysis (with correlation matrix) of the 200 most frequent words, and visualize the samples position in the space defined by the first two principal components, users can issue the following commands:

```
stylo(corpus.dir = "directory_containing_the_files", mfw.min = 200, mfw.max = 200,
       analysis.type = "PCR", sampling = "normal.sampling", sample.size = 10000,
       gui = FALSE)
```

In Fig. 1, we give an example of how Principal Components Analysis (the first two dimensions) can be used to visualize texts in different ways, e.g. with and without feature loadings. Because researchers are often interested in inspecting the loadings of features in the first two components resulting from such an analysis, **stylo** provides a rich variety of flavours in PCA visualizations. For an experiment in the domain of authorship studies, for instance, researchers will typically find it useful to plot all texts/samples from the same author in the same color. The coloring of the items in plots can be easily controlled via the titles of the texts analyzed across the different R methods that are used for visualization – a commodity which is normally rather painful to implement across different packages in R. Apart from exploratory, unsupervised analyses, **stylo** offers a number of classification routines that will be discussed below.

The examples shown in Fig. 1 were produced using the following functions:

```
stylo(frequencies = culled.freqs, analysis.type = "PCR",
        custom.graph.title = "Austen vs. the Bronte sisters",
        pca.visual.flavour = "technical",
        write.png.file = TRUE, gui = FALSE)

stylo(frequencies = culled.freqs, analysis.type = "PCR",
        custom.graph.title = "Austen vs. the Bronte sisters",
        write.png.file = TRUE, gui = FALSE)

stylo(frequencies = culled.freqs, analysis.type = "PCR",
        custom.graph.title = "Austen vs. the Bronte sisters",
        pca.visual.flavour = "symbols", colors.on.graphs = "black",
        write.png.file = TRUE, gui = FALSE)

stylo(frequencies = culled.freqs, analysis.type = "PCR",
        custom.graph.title = "Austen vs. the Bronte sisters",
        pca.visual.flavour = "loadings",
        write.png.file = TRUE, gui = FALSE)
```

Return value

Stylo makes it easy to further process the objects returned by an analysis. To cater for the needs of less technical users, the results returned by an analysis are saved by default to a number of standard files and outputted on screen. Advanced users can easily use the returned objects in subsequent processing:

```
stylo.results = stylo()  # optional arguments might be passed

print(stylo.results)
summary(stylo.results)
```

The list of features created, for instance, can be easily accessed (and manipulated) subsequently, and the same applies to tables of frequencies or other results:

```
stylo.results$features
stylo.results$table.with.all.freqs
stylo.results$distance.table
stylo.results$pca.coordinates
```

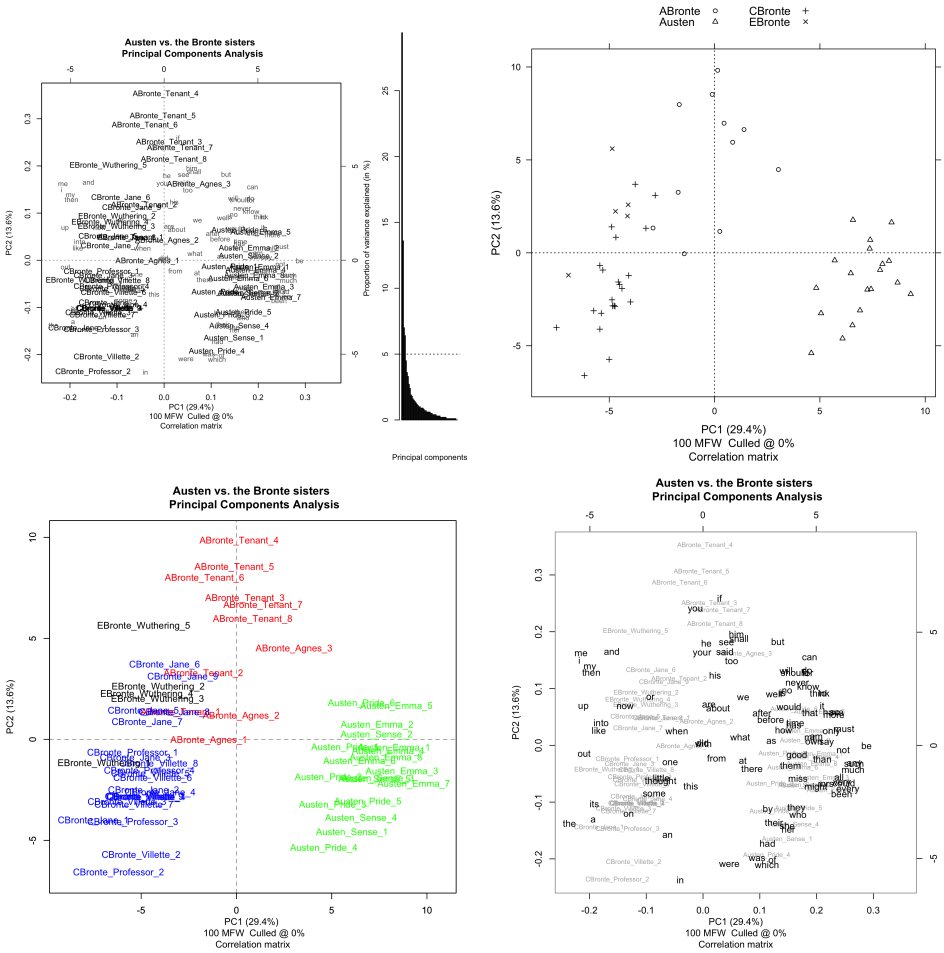


Figure 1: Illustration of different visualization options for the first two dimensions outputted by a Principal Components Analysis (applied to 9 novels by 4 authors from our dummy corpus). Four different visualization flavours are presented: 'Technical' (Fig. 1a), 'Classic' (Fig. 1b), 'Symbols' (Fig. 1c) and 'Loadings' (Fig. 1d). Users whose file names follow **stylo**'s naming conventions can easily exploit different coloring options.

GUI mode

Apart from the various functions to perform actual stylometric tasks, the package comes with a series of GUIs that can be used to set up typical experimental workflows in a quick and intuitive fashion. This unique feature renders **stylo** especially useful in educational settings involving students and scholars without programming experience. The cross-platform graphical user interface (automatically installed along with the rest of the package) has been written for Tcl/Tk and can be easily invoked from the command line. Four GUIs are currently available, which all come with extensive tooltips to help users navigate the different options. In this section, we will illustrate the use of these GUIs via an unsupervised stylometric experiment involving Bootstrap Consensus Trees.

The currently most widely used GUI component of 'Stylometry with R' is the eponymous GUI for `stylo()`, which is useful for the unsupervised stylistic exploration of textual corpora. It can be easily invoked using a single intuitive command (without the need to specify additional arguments):

```
stylo()
```

The various tabs of the **stylo** GUI (see Figure 2) present in a clear fashion the various parameters which can be specified before running the analysis by clicking the `OK` button. Users can freely switch between tabs and revisit them before running an experiment. Moreover, **stylo** will remember the experimental settings last used, and automatically default to these when users re-launch the GUI (which is useful for authors running a series of consecutive experiments with only small changes in parameters).

To illustrate the GUI mode, we will now concisely discuss a sample experiment involving Bootstrap Consensus Trees (BCT, selectable under the STATISTICS tab in the GUI). In stylometry, BCT exploits the

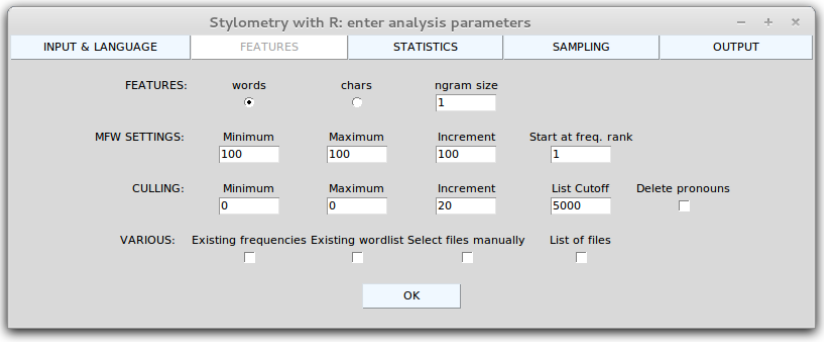


Figure 2: The Graphical User Interface for **stylo** (FEATURES tab). The high-level functions of **stylo** provide Graphical User Interfaces (GUIs) to intuitively set up experimental workflows. This feature is especially useful in an educational setting, allowing students without programming experience to engage in stylometric exploration and experimentation.

idea that the results become stable when one divides the list of MFW in non-identical, yet potentially overlapping frequency bands and analyzes these independently from each other (Eder, 2012). BCT were originally borrowed by Eder from the field of Language Evolution and Genetics; since a number of successful applications of the technique have been reported in the literature (Rybicki and Heydel, 2013; van Dalen-Oskam, 2014; Stover et al., 2016). If the user specifies that different frequency bands should be used on the FEATURES tab, the bootstrap procedure will run different (virtual) cluster analyses and aggregate the results into a single (unrooted) consensus tree. This visualization will only consider nodes for which there exists a sufficiently large consensus among the individual cluster analyses. The user in the corresponding text field (e.g. 0.5, which comes down to a majority vote for the cluster nodes). As such, users can assess the similarities between texts across different frequency bands.

Under the FEATURES tab, users can define the minutes of the MFW division and sampling procedure, using the `increment`, the `minimum` and `maximum` parameters. For `minimum = 100`, `maximum = 3000`, and `increment = 50`, stylo will run subsequent analyses for the following frequency bands: 100 MFW, 50–150 MFW, 100–200 MFW, ..., 2900–2950 MFW, 2950–3000 MFW. This is an attractive feature because it enables the assessment of similarities between texts across different bands in the frequency spectrum. A parallel logic underpins the CULLING text fields, where experiments will be carried out iteratively for different culling rates.

We illustrate the working of the BCT procedure in **stylo** using the recently covered case study on *Go Set a Watchman*, the second novel by Harper Lee, written before *To Kill a Mockingbird*. The novel itself attracted a reasonable attention worldwide, also because of its alleged authorship issues. Suspicion resurfaced about the strange fact that one of the greatest bestsellers in American history was its author's only completed work; Lee's childhood friendship with Truman Capote (portrayed as Dill in *To Kill A Mockingbird*) and their later association on the occasion of *In Cold Blood* fueled more speculations on the two Southern writers' possible, or even just plausible, collaboration; finally, the role of Tay Hohoff, Lee's editor on her bestseller, was discussed.

The stylometric study on this novel, featured in *Wall Street Journal* (Gamerman, 2015), revealed that the truth proved to be at once much less sensational than most of the rumors. Very strong stylometric evidence shows clearly that Harper Lee is the author of both *To Kill A Mockingbird* and *Go Set A Watchman*. In our replication of the experiment, the following code was used to produce the plots:

```
data(lee)

stylo(frequencies = lee, analysis.type = "CA",
        write.png.file = TRUE,  custom.graph.title = "Harper Lee",
        gui = FALSE)

stylo(frequencies = lee, analysis.type = "CA",
        mfw.min = 1500, mfw.max = 1500, custom.graph.title = "Harper Lee",
        write.png.file = TRUE, gui = FALSE)

stylo(frequencies = lee, analysis.type = "BCT",
        mfw.min = 100, mfw.max = 3000, custom.graph.title = "Harper Lee",
        write.png.file = TRUE, gui = FALSE)
```

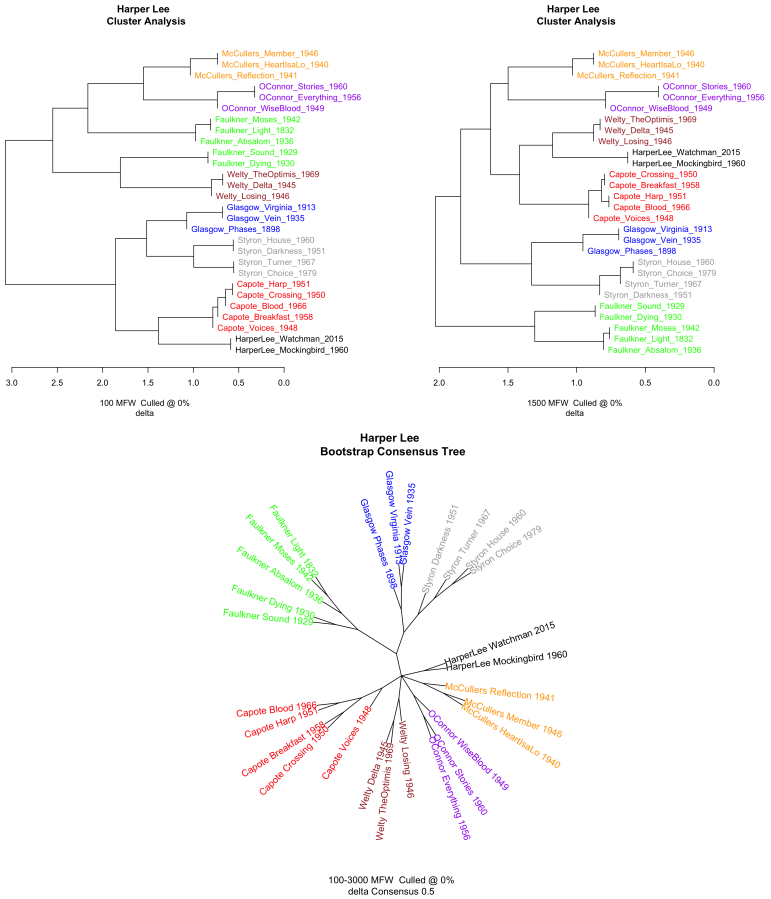


Figure 3: Analysis of the corpus of 28 novels by Harper Lee, Truman Capote as well as a number of comparable control authors writing in the American South. A frequency table of this corpus is provided by the package **stylo**, so that all our experiments can be replicated. In all plots, Lee's writing style is clearly very consistent, even if for some input parameters Lee's novels are close to Capote's. Figure panel 3a-3b: Traditional dendrograms outputted by cluster analyses with Burrows's Classic Delta Metric for 100 MFW and 1,500 MFW respectively (default settings; entire novels). Figure panel 3c: Bootstrap consensus tree for 100 MFW to 3,000 MFW (with an incremental step size of 50 words). Unrooted tree which combines clade information from analyses such as the ones presented in Fig. 1a-1b. The tree collapses nodes which were observed in at least 50% of the underlying trees (majority vote).

Classify

Apart from the already-discussed explanatory multivariate tests and the associated visualizations, stylometry has borrowed a number of advanced classification methods from the domain of Machine Learning. Some of them have simply been transferred to stylometry (e.g. Support Vector Machines or Naïve Bayes Classifier); others have been tailored to the needs of humanities researchers. The best example in this respect is Delta, a so-called 'lazy' learner developed by Burrows (Burrows, 2002). The **stylo** package offers an interface to a selection of established classifiers: including Burrows's original Delta and other distance-based classifiers, Nearest Shrunken Centroids, Support Vector Machines and Naïve Bayes Classifier. These are available through a single function:

```
classify()   # optional arguments might be passed
```

If any non-standard text preprocessing procedures are involved, the above function can be fed with the result of a multi-stage custom pipeline. Combining the function `classify()` with spreadsheet tables of frequencies is also possible.

In a typical classification experiment, the analysis is divided in two stages. In the first stage, representative text samples for each target category (e.g. authorial group) are collected in a training corpus. The remaining samples form the test corpus. The first set, being a collection of texts, e.g. written by known authors ('candidates'), serves as a sub-corpus for fine-tuning the hyperparameters of a classifier and model architecture selection. The second set is a pool that consists of test texts of known

authorship and anonymous texts of disputed authorial provenance. The classifier's performance can be measured by applying a standard evaluation metric to the classifier's output on the test set (e.g. the number of correct attributions to authors in the the training set). In **stylo**, users can divide their data over two subdirectories (or input custom-created R objects using the low-level functions discussed above); one directory should contain the training samples, the other the test samples. Other options can be specified via the parameters that run parallel to those of the `stylo()` function, such as the desired feature type or culling rate. Function-specific parameters for `classify()` include the number of cross-validation folds or the type of classifier (e.g. Support Vector Machine).

We illustrate the performance of classification methods in **stylo** using the well-known case study of the pseudonymous author Galbraith/Rowling, which recently attracted a good deal of press attention. In July 2013, the *Sunday Times* (UK) revealed that J. K. Rowling, the successful author behind the bestselling series of *Harry Potter* novels, had published a new detective novel (*The Cuckoo's Calling*) under the pseudonym of 'Robert Galbraith'. (The paper had received an anonymous tip with respect to this pen name over Twitter). For covering this case study, the *Sunday Times* has collaborated with Patrick Juola, an authority in the field of authorship attribution, and Peter Millican (Juola, 2013). They reported in a blog post on the Language Log that their stylometric analysis showed the writing style (e.g. on the level of function words) found in *The Cuckoo's Calling* to be broadly consistent with Rowling's writing in other works. Below, we report on a dummy attribution experiment which illustrates a supervised procedure.

In this experiment we will confront Galbraith's *The Cuckoo's Calling* with 25 other fantasy novels and thrillers by 4 famous novelists: H. Coben (e.g. *Tell No One*), C. S. Lewis (e.g. *The Chronicles of Narnia*), J. R. R. Tolkien (e.g. the *Lord of the Rings* trilogy) and J. K. Rowling (e.g. the *Harry Potter* series). Our replication experiments indeed confirm that Galbraith's writing style is more consistent with that of Rowling than that of any other author included. Instead of loading particular text files, we will use a computed table of frequencies provided by the package; the table has to be split into two tables (training set and test set). As an illustration, we specify the training set manually (with two training texts per class):

```
# specify a table with frequencies:
data(galbraith)
freqs <- galbraith

# specify class labels:
training.texts <- c("coben_breaker", "coben_dropshot", "lewis_battle",
                    "lewis_caspian", "rowling_casual", "rowling_chamber",
                    "tolkien_lord1", "tolkien_lord2")

# select the training samples:
training.set <- freqs[(rownames(freqs) %in% training.texts),]

# select remaining rows as test samples:
test.set <- freqs[!(rownames(freqs) %in% training.texts),]
```

To perform Delta on the Rowling corpus (50 MFWs, no sampling), we type:

```
classify(training.frequencies = training.set, test.frequencies = test.set,
       mfw.min = 50, mfw.max = 50, classification.method = "delta",
       gui = FALSE)
```

The results are automatically outputted to a log file 'final_results.txt':

```
galbraith_cuckoos   -->   rowling rowling coben

50 MFW, culled @ 0%,  17 of 17        (100%)

General attributive success:  17 of 17 (100%)

MFWs from 50 to 50 @ increment 100
Culling from 0 to 0 @ increment 20
Pronouns deleted: FALSE; standard classification
```

The overall performance of the classifier for our dummy corpus is optimal, since 100% of the test samples were correctly attributed to the correct authors. The experiment adds support to the identification of the author of *The Cuckoo's Calling* as Rowling. To combat model overfitting, cross-validation on the training data can be applied. It has been shown that for linguistic datasets a standard

10-fold cross validation might overestimate the performance of models, especially if languages other than English are assessed (Eder and Rybicki, 2013). To neutralize class imbalance, **stylo** therefore provides stratified cross-validation protocols for stylometric experiments. To perform a classification with a 'plain vanilla' 20-fold CV, using Nearest Shrunken Centroids classification and a series of tests for 50, 100, 150, 200, ..., 500 MFWs, one might type:

```
results <- classify(training.frequencies = training.set,
                    test.frequencies = test.set,
                    mfw.min = 50, mfw.max = 500, mfw.incr = 50,
                    classification.method = "nsc", cv.folds = 20, gui = FALSE)
```

To inspect the classification accuracy for particular cross-validation folds, the user can type:

```
results$cross.validation.summary
```

Average scores of the cross-validation outcome (note that the overall performance is now slightly worse, ca. 95%) can be accessed via:

```
colMeans(results$cross.validation.summary)
```

Miscellaneous other functions

Apart from the above discussed functions, the package offers miscellaneous other, less established functions to stylometrically analyze documents. With the `oppose()` function, users can contrast two sets of documents and extract the most characteristic features in both sets of texts. The most discriminative features can be visualized and fed into other components of the package as part of a pipeline. Several metrics are implemented that can select features which display a statistically significant difference in distributions between both sets. Craig's Zeta, for instance, is an extension of the Zeta metric originally proposed by Burrows (Burrows, 2007), which remains a popular choice in the stylometric community to select discriminative stylometric features in binary classification settings (Craig and Kinney, 2009). An example of another more widely used metric for feature selection in corpus linguistics is the Mann-Whitney ranks test (Kilgariff, 2001). As a dummy example, we can confront the above mentioned texts; be it the novels by Jane Austen and Anne Brontë:

```
data(novels)

corpus.all <- txt.to.words.ext(novels, language = "English.all",
                preserve.case = TRUE)

corpus.austen <- corpus.all[grep("Austen", names(corpus.all))]
corpus.abronte <- corpus.all[grep("ABronte", names(corpus.all))]

zeta.results <- oppose(primary.corpus = corpus.austen,
               secondary.corpus = corpus.abronte, gui = FALSE)
```

As can be seen in the results (first 20 most discriminating words), Jane Austen is an enthusiast user of terms related to socio-cultural phenomena (e.g. *situation, opinion, party, engaged, ...*), whereas Anne Brontë's vocabulary can be characterized by a variety of auxiliary verbs with contractions, as well as religious and light-related vocabulary (e.g. *bright, dark*).

```
zeta.results$words.preferred[1:20]

 [1] "Her"       "farther"    "behaviour"  "opinion"    "party"
 [6] "point"     "perfectly"  "afterwards" "Colonel"    "directly"
[11] "spirits"   "situation"  "settled"    "hardly"     "Jane"
[16] "Emma"      "equal"      "family"     "engaged"    "They"

zeta.results$words.avoided[1:20]

 [1] "don^t"     "I^m"        "I^ll"       "beside"     "Arthur"
 [6] "can^t"     "I^ve"       "it^s"       "won^t"      "Huntingdon"
[11] "presence"  "Helen"      "face"       "bright"     "God"
[16] "mamma"     "further"    "heaven"     "dark"       "feet"
```

Of course, the above results of this simple feature selection tool can be fed into one of the package's classification routines:

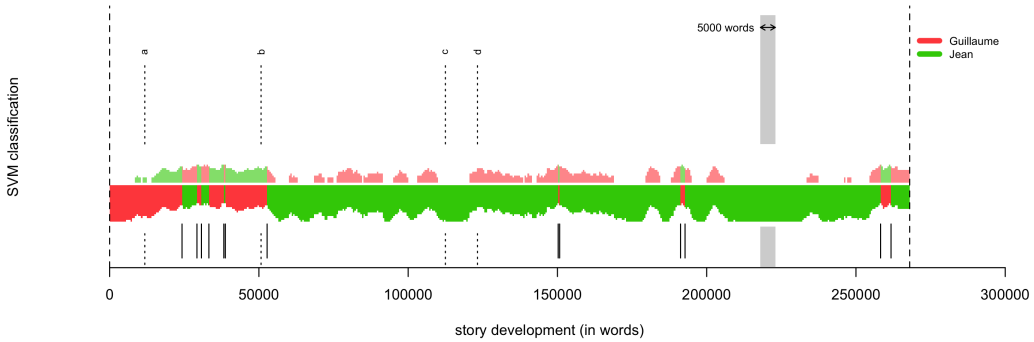


Figure 4: The Rolling Stylometry visualization. The medieval French allegoric story *Roman de la Rose* assessed using Rolling SVM and 100 MFWs; window size: 5,000 words, sample overlap: 4,500 words. Sections attributed to Guillaume de Lorris are marked red, those attributed to Jean de Meun are green. The level of certainty of the classification is indicated by the thickness of the bottom stripe. The commonly-accepted division into two authorial parts is marked with a vertical dashed line 'b'.

```
combined.features <- c(zeta.results$words.preferred[1:20],
                       zeta.results$words.avoided[1:20])
stylo(parsed.corpus = corpus.all, features = combined.features, gui = FALSE)
```

Other functionality worth mentioning are `rolling.delta()` and `rolling.classify()`. These functions implement a procedure meant to progressively analyze the development of a style in a text, using e.g. one of the stylometric distance metrics discussed (Rybicki et al., 2014; Eder, 2016). In many works, specific parts of the text are conjectured to have been plagiarized or contributed by other authors: `rolling.delta()` and `rolling.classify()` offer an easy way to visualize local stylistic idiosyncrasies in texts. In Fig. 4 we have plotted a `rolling.classify()` analysis of the well-known French allegorical romance *Roman de la Rose* from the Middle Ages. It has been written by two authors: Guillaume de Lorris is the author of the opening 4,058 lines (ca. 50,000 words), and the second part by Jean de Meun consists of 17,724 lines (ca. 218,000 words). This knowledge is supported by the text itself, since Jean de Meun explicitly points out the takeover point (it is marked with a dashed vertical line 'b' in Fig. 4). In this example, the aim is to verify whether two authorial styles can indeed be discerned in the text, that is, before and after the authorial takeover. First a Support Vector Machine classifier is trained on four 5,000-word samples: two extracted from the beginning of the text and two near the middle of the text (yet well beyond the hypothesized takeover: they are marked with the dashed line 'a' and 'c–d', respectively). Next, we apply a windowing procedure and we extract consecutive and partially overlapping samples from the entire text. Finally, the trained classifier is applied to each of these 'windows.' In Fig. 4 we plot the respective classification scores for both authors in each sample: in this case, these scores represent the probability, estimated by a Support Vector Machine, that a particular sample should be attributed to one of the two authors involved. Although the result is not flawless, a clear shift in authorial style can be discerned around the position of the takeover, as indicated verbatimly in the text by one the authors.

The dataset to replicate the test can be downloaded from this page: https://sites.google.com/site/computationalstylistics/corpora/Roman_de_la_Rose.zip. The following code should be typed to perform the classification:

```
# unzipping the dataset
unzip("Roman_de_la_Rose.zip")

# changing working directory
setwd("Roman_de_la_Rose")

rolling.classify(write.png.file = TRUE, classification.method = "svm", mfw = 100,
        training.set.sampling = "normal.sampling", slice.size = 5000,
        slice.overlap = 4500)
```

Conclusion

'Stylometry with R' targets two distinct groups of users: experienced coders and beginners. Novice users have found it useful to work with the intuitive Graphical User Interface (GUI), which makes it easy to set and explore different parameters without programming experience. We wish to emphasize, however, that **stylo** is useful beyond these high-level functions and GUIs: it also offers experienced users a general framework that can be used to design custom processing pipelines in R, e.g. in other text-oriented research efforts. The current version of **stylo** (version number 0.6.3) is available from GitHub under a GPL 3.0 open-source licence; binary installation files are available from CRAN. **stylo** has been used in a number of innovative studies in the field of computational stylistics (Kestemont et al., 2013; van Dalen-Oskam, 2014; Lauer and Jannidis, 2014; Anand et al., 2014; Oakes and Pichler, 2013; Boot, 2013), and we encourage the future application of **stylo** to challenging new problems and languages in stylometry.

Acknowledgments

We would like to thank the users of **stylo** for the valuable feedback and feature requests which we have received over the past years. MK was partially founded for this research as a postdoctoral fellow by The Research Foundation of Flanders (FWO). ME was partially supported by Poland's National Science Centre (grant number 2014/12/W/ST5/00592).

Bibliography

S. Anand, A. K. Dawn, and S. K. Saha. A statistical analysis approach to author identification using latent semantic analysis. notebook for PAN at CLEF 2014. In *CLEF2014 Working Notes*, pages 1143–1147, Sheffield, UK, 2014. CLEF. [p119]

S. Argamon. Interpreting Burrows's Delta: Geometric and probabilistic foundations. *Literary and Linguistic Computing*, 23(2):131–147, 2011. [p109]

P. Boot. Online boekdiscussie van een afstand gelezen. *TNTL. Journal of Dutch Linguistics and Literature*, 129(4):215–232, 2013. [p119]

J. Burrows. 'Delta': A measure of stylistic difference and a guide to likely authorship. *Literary and Linguistic Computing*, 17(3):267–287, 2002. [p109, 115]

J. Burrows. All the way through: testing for authorship in different frequency strata. *Literary and Linguistic Computing*, 22(1):27–48, 2007. [p117]

H. Craig and A. Kinney, editors. *Shakespeare, Computers, and the Mystery of Authorship*. Cambridge University Press, 2009. [p117]

W. Daelemans. Explanation in computational stylometry. In *Proceedings of the 14th International Conference on Computational Linguistics and Intelligent Text Processing – Volume 2*, CICLing'13, pages 451–462, Berlin, Heidelberg, 2013. Springer-Verlag. ISBN 978-3-642-37255-1. [p108]

M. Eder. Style-markers in authorship attribution: a cross-language study of the authorial fingerprint. *Studies in Polish Linguistics*, 6:99–114, 2011. URL http://www.wuj.pl/page,art,artid,1923.html. [p111]

M. Eder. Computational stylistics and biblical translation: how reliable can a dendrogram be? In T. Piotrowski and Ł. Grabowski, editors, *The Translator and the Computer*, pages 155–170. WSF Press, Wrocław, 2012. [p114]

M. Eder. Rolling stylometry. *Digital Scholarship in the Humanities*, 31(3):457–469, 2016. [p118]

M. Eder and J. Rybicki. Do birds of a feather really flock together, or how to choose training samples for authorship attribution. *Literary and Linguistic Computing*, 28(2):229–236, 2013. [p117]

I. Feinerer, K. Hornik, and D. Meyer. Text mining infrastructure in R. *Journal of Statistical Software*, 25 (5):1–54, 3 2008. ISSN 1548-7660. URL http://www.jstatsoft.org/v25/i05. [p108]

E. Gamerman. Data miners dig into 'Watchman'. *Wall Street Journal*, page D5, July 2015. [p107, 114]

D. Holmes. Authorship attribution. *Computers and the Humanities*, 28(2):87–106, 1994. [p107]

D. Holmes. The evolution of stylometry in Humanities scholarship. *Literary and Linguistic Computing*, 13(3):111–117, 1998. [p107]

D. Hoover. Testing Burrows's Delta. *Literary and Linguistic Computing*, 19(4):453–475, 2004a. [p109, 110]

D. Hoover. Delta prime. *Literary and Linguistic Computing*, 19(4):477–495, 2004b. [p109, 110]

J. Houvardas and E. Stamatatos. N-gram feature selection for authorship identification. In J. Euzenat and J. Domingue, editors, *Proceedings of Artificial Intelligence: Methodologies, Systems, and Applications*, pages 77–86. Springer Verlag, 2006. [p108]

M. Jockers. *Macroanalysis: Digital Methods and Literary History*. Topics in the Digital Humanities. University of Illinois Press, 2013. ISBN 9780252094767. [p108]

P. Juola. Authorship attribution. *Foundations and Trends in Information Retrieval*, 1(3):233–334, 2006. [p107]

P. Juola. Becoming Jack London. *Journal of Quantitative Linguistics*, 14(2):145–147, 2007. [p108]

P. Juola. Rowling and "Galbraith": an authorial analysis, 2013. URL http://languagelog.ldc.upenn.edu/nll/?p=5315. [p107, 116]

M. Kestemont. Function words in authorship attribution: From black magic to theory? In *Proceedings of the 3rd Workshop on Computational Linguistics for Literature (CLFL)*, pages 59–66, Gothenburg, Sweden, April 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W14-0908. [p108]

M. Kestemont, S. Moens, and J. Deploige. Collaborative authorship in the twelfth century: A stylometric study of Hildegard of Bingen and Guibert of Gembloux. *Literary and Linguistic Computing*, 28: 1–15, 2013. doi: doi:10.1093/llc/fqt063. [p119]

A. Kilgariff. Comparing corpora. *International Journal of Corpus Linguistics*, 6(1):97–133, 2001. [p117]

B. Kjell. Discrimination of authorship using visualization. *Information Processing and Management*, 30 (1):141–50, 1994. [p108, 111]

M. Koppel, J. Schler, and S. Argamon. Computational methods in authorship attribution. *Journal of the American Society for Information Science and Technology*, 60(1):9–26, 2009. [p107]

G. Lauer and F. Jannidis. Burrows's delta and its use in German literary history. In M. Erlin and L. Tatlock, editors, *Distant Readings. Topologies of German Culture in the Long Nineteenth Century*, pages 29–54. Rochester, New York, 2014. [p119]

H. Love. *Attributing authorship. An introduction*. Cambridge University Press, Cambridge, 2002. [p107]

M. Oakes and A. Pichler. Computational stylometry of Wittgenstein's 'Diktat für Schlick'. *Bergen Language and Linguistics Studies*, 3(1):221–240, 2013. [p119]

J. Pennebaker. *The Secret Life of Pronouns: What our Words Say about Us*. Bloomsbury Press, New York, 2011. [p110]

J. Rybicki and M. Heydel. The stylistics and stylometry of collaborative translation: Woolf's 'Night and Day' in Polish. *Literary and Linguistic Computing*, 28(4):708–717, 2013. [p114]

J. Rybicki, D. Hoover, and M. Kestemont. Collaborative authorship: Conrad, Ford and rolling delta. *Literary and Linguistic Computing*, 29(3):422–431, 2014. [p118]

S. Schreibman, R. Siemens, and J. Unsworth, editors. *A companion to Digital Humanities*. Blackwell, 2004. URL http://www.digitalhumanities.org/companion/. [p107]

F. Sebastiani. Machine Learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002. [p108]

E. Stamatatos. A survey of modern authorship attribution methods. *Journal of the American Society for Information Science and Technology*, 60(3):538–556, 2009. [p107, 108, 111]

E. Stamatatos, N. Fakotakis, and G. Kokkinakis. Automatic text categorization in terms of genre and author. *Computational Linguistics*, 26(4):471–495, 2000. [p107]

C. Stamou. Stylochronometry: stylistic development, sequence of composition, and relative dating. *Literary and Linguistic Computing*, 23(2):181–199, 2008. [p107]

J. Stover, Y. Winter, M. Koppel, and M. Kestemont. Computational authorship verification method attributes new work to major 2nd century African author. *Journal of the American Society for Information Science and Technology*, 67(1):239–242, 2016. [p114]

K. van Dalen-Oskam. Epistolary voices: The case of Elisabeth Wolff and Agatha Deken. *Literary and Linguistic Computing*, 29(3):443–451, 2014. [p114, 119]

L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *The Journal of Machine Learning Research*, 9(2579–2605):85, 2008. [p111]

H. Van Halteren, H. Baayen, F. Tweedie, M. Haverkort, and A. Neijt. New machine learning methods demonstrate the existence of a human stylome. *Journal of Quantitative Linguistics*, 12(1):65–77, 2005. [p107]

Maciej Eder
Institute of Polish Language
Polish Academy of Sciences
al. Mickiewicza 31, 31-120 Kraków
Poland
maciejeder@gmail.com

Jan Rybicki
Institute of English Studies
Jagiellonian University
al. Mickiewicza 9A, 31-120 Kraków
Poland
jkrybicki@gmail.com

Mike Kestemont
Department of Literature
University of Antwerp
Prinsstraat 13, B-2000 Antwerp
Belgium
mike.kestemont@uantwerp.be

quickpsy: An R Package to Fit Psychometric Functions for Multiple Groups

by Daniel Linares and Joan López-Moliner

Abstract **quickpsy** is a package to parametrically fit psychometric functions. In comparison with previous R packages, **quickpsy** was built to easily fit and plot data for multiple groups. Here, we describe the standard parametric model used to fit psychometric functions and the standard estimation of its parameters using maximum likelihood. We also provide examples of usage of **quickpsy**, including how allowing the lapse rate to vary can sometimes eliminate the bias in parameter estimation, but not in general. Finally, we describe some implementation details, such as how to avoid the problems associated to round-off errors in the maximisation of the likelihood or the use of closures and non-standard evaluation functions.

Introduction

Statistical model

The response of humans, other animals and neurons in a classification task with a binary response variable and a stimulus level as explanatory variable is often binomially modelled as (Watson, 1979; O'Regan and Humbert, 1989; Klein, 2001; Wichmann and Hill, 2001a; Macmillan and Creelman, 2004; Gold and Shadlen, 2007; Kingdom and Prins, 2009; Knoblauch and Maloney, 2012; Lu and Dosher, 2013; Gold and Ding, 2013)

$$f(\mathbf{k};\boldsymbol{\theta}) = \prod_{i=1}^{M} \binom{n_i}{k_i} \psi(x_i;\boldsymbol{\theta})^{k_i} \left(1-\psi\left(x_i;\boldsymbol{\theta}\right)\right)^{n_i-k_i}, \tag{1}$$

where

- f is the probability mass function of the model or the likelihood when considered as a function of the parameters;
- M is the number of stimulus levels used in the classification task;
- x_i is the i*th* stimulus level;
- n_i is the number of times that x_i is presented;
- $\mathbf{k} = (k_1, k_2, \ldots, k_M)$ is the vector of responses with k_i being the number of *Yes-type* (or correct) responses when x_i is presented;
- $\psi(x_i;\boldsymbol{\theta})$ is the probability of responding *Yes* when x_i is presented; it is called the *psychometric function* and has the form

$$\psi(x;\boldsymbol{\theta}) = \psi(x;\alpha,\beta,\gamma,\lambda) = \gamma + (1-\gamma-\lambda)F(x;\alpha,\beta), \tag{2}$$

  where

  - $\boldsymbol{\theta} = (\alpha,\beta,\gamma,\lambda)$ is the vector of parameters that define the parametric family of probability mass functions of the model. α and β are the position and scale parameters. γ and λ are the parameters corresponding to the leftward and rightward asymptote of ψ.
  - F is a function with leftward asymptote 0 and rightward asymptote 1—typically a cumulative probability function with a sigmoidal shape such as the cumulative normal, logistic or Weibull functions.

The model assumes that a given classification response does not depend on previous classifications. This is an idealisation, given the known order effects such as adaptation, fatigue, learning or serial dependence (Kingdom and Prins, 2009; Fründ et al., 2011; Van der Burg et al., 2013; Fischer and Whitney, 2014; Summerfield and Tsetsos, 2015).

Examples

Light detection. To measure the ability of an observer to detect light, a dim flash of light selected at random from 5 different light intensities (x_i; $i = 1,\ldots,M$ with $M = 5$) is presented and the observer is

asked to report *Yes* if she has seen it and *No* otherwise. After her response, another intensity, selected at random from the 4 intensities that have not been presented yet, is presented and the observer classifies it again as seen or not seen. Then, the observer performs 3 more classifications until the 5 intensities have been presented. After that, the whole procedure is repeated 20 times using a new random sequence of 5 intensities each time. Using this procedure, which is called the method of constant stimuli (Green and Swets, 1966; Gescheider, 1997; Kingdom and Prins, 2009; Knoblauch and Maloney, 2012), the observer will perform a total of 100 classifications with $n_i = 20$ for all i. k_i will correspond to the number of times that the observer responds *Yes* for each intensity. Because it is expected that, for very low intensities, the observer will never respond *Yes* and for very high intensities the observer will always respond *Yes*, γ and λ are often fixed to 0.

Criterion-independent light detection. In the previous procedure, $\mathbf{k}$ depends on how confident the observer needs to feel to give a *Yes* response—conservative observers will repond *Yes* less often (Green and Swets, 1966; Macmillan and Creelman, 2004; Kingdom and Prins, 2009; Knoblauch and Maloney, 2012; Lu and Dosher, 2013). To avoid criterion-dependent responses, 2-intervals forced choice procedures are often used (Green and Swets, 1966; Kingdom and Prins, 2009; Knoblauch and Maloney, 2012; Lu and Dosher, 2013). These procedures are similar to the criterion-dependent procedure described above, but the stimulus is presented at random in one interval from two intervals presented consecutively (marked with a sound, for example) and the observer needs to decide whether the stimulus was presented in the first or the second interval. Because it is expected that for very low intensities the observer will respond at chance, γ is fixed at $1/2$ (λ is usually fixed to 0). More generally, γ is fixed to $1/m$ when the observer needs to decide in which over m intervals the stimulus was presented.

Light detection with lapses. Sometimes, the observer will miss the flash (because of a blink, for example) or will make an error reporting the response (pressing the wrong response button, for example). To account for these response *lapses*, λ, which corresponds to $(1-\gamma)$ times the lapse rate (Kingdom and Prins, 2009), is not fixed but estimated as a parameter (Wichmann and Hill, 2001a,b).

Point estimation and confidence intervals

The *point estimation* of the parameters $\boldsymbol{\theta}$ of the model in (1), $\hat{\boldsymbol{\theta}}$, is sometimes obtained using Bayesian methods (Kuss et al., 2005; Kingdom and Prins, 2009), but more often using maximum likelihood (ML; Watson, 1979; O'Regan and Humbert, 1989; Wichmann and Hill, 2001a; Kingdom and Prins, 2009; Knoblauch and Maloney, 2012; Lu and Dosher, 2013) . $\hat{\boldsymbol{\theta}}$ is defined as the value of $\boldsymbol{\theta}$ that maximises the likelihood L defined as $L(\boldsymbol{\theta}) = f(\mathbf{k}; \boldsymbol{\theta})$.

Maximising L is equivalent to maximising $\log(L)$, which for the model in (1) is

$$\log L(\boldsymbol{\theta}) = \sum_{i=1}^{M} \left(\log \binom{n_i}{k_i} + k_i \log \psi\left(x_i; \boldsymbol{\theta}\right) + \left(n_i - k_i\right) \log\left(1 - \psi\left(x_i; \boldsymbol{\theta}\right)\right) \right). \tag{3}$$

The *confidence intervals* CI for $\boldsymbol{\theta}$ are usually estimated using parametric or non-parametric bootstrap, often using the percentile method (Wichmann and Hill, 2001b; Kingdom and Prins, 2009; Knoblauch and Maloney, 2012). For example, for the parameter α, the bootstrap percentile interval is defined as $(\alpha^*_{(a/2)}, \alpha^*_{(1-a/2)})$, where $\alpha^*_{(a/2)}$ and $\alpha^*_{(1-a/2)}$ are the $a/2$ and the $1-a/2$ percentiles of the bootstrapped replications of $\hat{\alpha}$. The i*th* bootstrap replication α^*_i is obtained using ML, but this time for a vector of simulated responses $\mathbf{k}^*$. Each k^*_i is simulated from a binomial distribution with parameters n_i and p_i where p_i is $\psi(x_i; \hat{\boldsymbol{\theta}})$ for the parametric bootstrap and k_i/n_i for the non-parametric bootstrap (which is equivalent to sampling with replacement from the distribution of *Yes* and *No* responses). It could be demonstrated that $CI = (\alpha^*_{(a/2)}, \alpha^*_{(1-a/2)})$ is a well-defined confidence interval (Wasserman, 2013). That is, $P(\alpha \in CI) \geq 1 - a$ where P is a probability and a is often arbitrarily chosen as 0.05. The confidence intervals for the other parameters are obtained similarly.

The observer's behaviour in classification tasks is often summarised by a *threshold* x_{th} (O'Regan and Humbert, 1989; Wichmann and Hill, 2001a; Kingdom and Prins, 2009; Knoblauch and Maloney, 2012; Lu and Dosher, 2013), which corresponds to the stimulus level that predicts an arbitrarily chosen proportion of *Yes* responses. If the proportion is 0.5, for example, then x_{th} would be the x for which $\psi(x; \hat{\boldsymbol{\theta}}) = 0.5$. The bootstrap CIs for x_{th} can be obtained using the percentile method for x^*_{th}, which are the bootstrap replications of x_{th} calculated using the bootstrap replications of the parameters.

ML estimates of the parameters are often obtained using non-linear optimisation (Nash, 2014), a method that might produce unsuitable estimates when the data suffer from lapses as those described above. Future studies might elucidate the possible problems of non-linear optimisation to fit psychometric functions with lapses and whether Bayesian approaches might be preferred (Kuss et al., 2005).

quickpsy and similar tools

quickpsy (Linares and López-Moliner, 2016) is a package to estimate and plot the parameters, the thresholds, the bootstrap confidence intervals for the parameters and the thresholds, and the associated psychometric functions for the model in (1). **quickpsy** also allows the comparison of the parameters and the thresholds for different groups using the bootstrap. It is build to easily analyse data for multiple groups, which is common in classification experiments (Gescheider, 1997; Lu and Dosher, 2013): multiple observers, for example, might be tested under different conditions, such as flashes of different size in the light detection experiments. For that purpose, **quickpsy** incorporates, for example, a function to easily create a data frame from multiple files (`quickreadfiles`) or uses the dimensions of the groups in the data to produce standard plots for the parameters, the thresholds and the associated psychometric functions.

As an alternative to **quickpsy**, classification data can also be analysed in R using the base function `glm` for fitting generalized linear models and tools from package **psyphy** (Knoblauch and Maloney, 2012; Knoblauch, 2014). Fitting psychometric functions could be considered a special case of fitting *generalised linear models (GLM)* (Knoblauch and Maloney, 2012; Moscatelli et al., 2012), which is done in R using `glm` (Knoblauch and Maloney, 2012). With `glm`, one can estimate the parameters of the linear predictor associated with the GLM and use them to estimate the parameters of the model in (1) (Knoblauch and Maloney, 2012). Also by applying `confint` to the `glm` output (Knoblauch and Maloney, 2012), the confidence intervals can be calculated using the profile likelihood method (Venables and Ripley, 2002). To calculate the parameters and confidence intervals for multiple groups, the user needs to manually or automatically *loop* by group. Alternatively, if one is not interested in the individual values of the parameters for each group, but on fitting a single model to the multiple groups, `glm` allows to do that using advanced statistical methods (Knoblauch and Maloney, 2012; Moscatelli et al., 2012).

To estimate the parameters in (1) using `glm` is straightforward when $\gamma = 0$ and $\lambda = 0$ (to see `glm` in action for the `HSP` dataset, see Knoblauch and Maloney 2012), but becomes more complicated when γ and λ are not 0 or need to be estimated (Knoblauch and Maloney, 2012). To facilitate the application of `glm` when λ is not zero, **psyphy** provides specialised link functions. Furthermore, **psyphy** includes the `psyfun.2asym` function, which by iterating `glm` calls, allows the estimation of γ and λ.

The specific shape of F in (1) is sometimes chosen based on some theory (Green and Swets, 1966; Quick, 1974; Kingdom and Prins, 2009). Other times, especially when one is only interested in calculating the threshold, it is chosen arbitrarily. In those cases, one might prefer to fit a non-parametric model to the data, which can be done in R using package **modelfree** (Zychaluk and Foster, 2009; Marin-Franch et al., 2012),

In other languages, the current available tools to calculate the parameters in (1) and its confidence intervals are **psignifit** (Fründ et al., 2011) for Python (free), **psycophysica** (Watson and Solomon, 1997) for Mathematica (commercial) and **palamedes** (Kingdom and Prins, 2009) for MATLAB (commercial). From them, **palamedes** can also fit models for multiple groups. Furthermore, **palamedes** and **psignifit** can fit psychometric functions using Bayesian methods, which is something not implemented in **quickpsy**.

Examples of usage

Light detection

A classic experiment on light detection (similar to the one first described in the Introduction) was conducted by Hecht et al. (1942). The data from the experiment is available in **MPDiR**, a package that includes material from the book *Modeling Psychophysical Data in R* (Knoblauch and Maloney, 2012).

The data is included in the data frame `HSP`, which has a *tidy* structure (Wickham, 2014), that is, each column corresponds to a variable and each row is an observational unit. The variables in `HSP` are the intensity level measured in quanta `Q` (what we named stimulus level x_i), the number of times that each intensity was presented `N` (what we named n_i) and two variables identifying the groups: the identifier of the observer `Obs` and the run for each observer `Run`. In the original dataset, the number of *Yes* responses for each stimulus level k_i was not given—the probability `p` of responding *Yes* was given instead. But, calculating the number of *Yes* responses from `p` is trivial:

```
library(MPDiR)
library(dplyr)
library(quickpsy)
HSP <- HSP %>% mutate(k = round(p * N / 100)) # adding a column with the number of Yes
```

We used the `round` function because, curiously, there was some error in the original data set (see

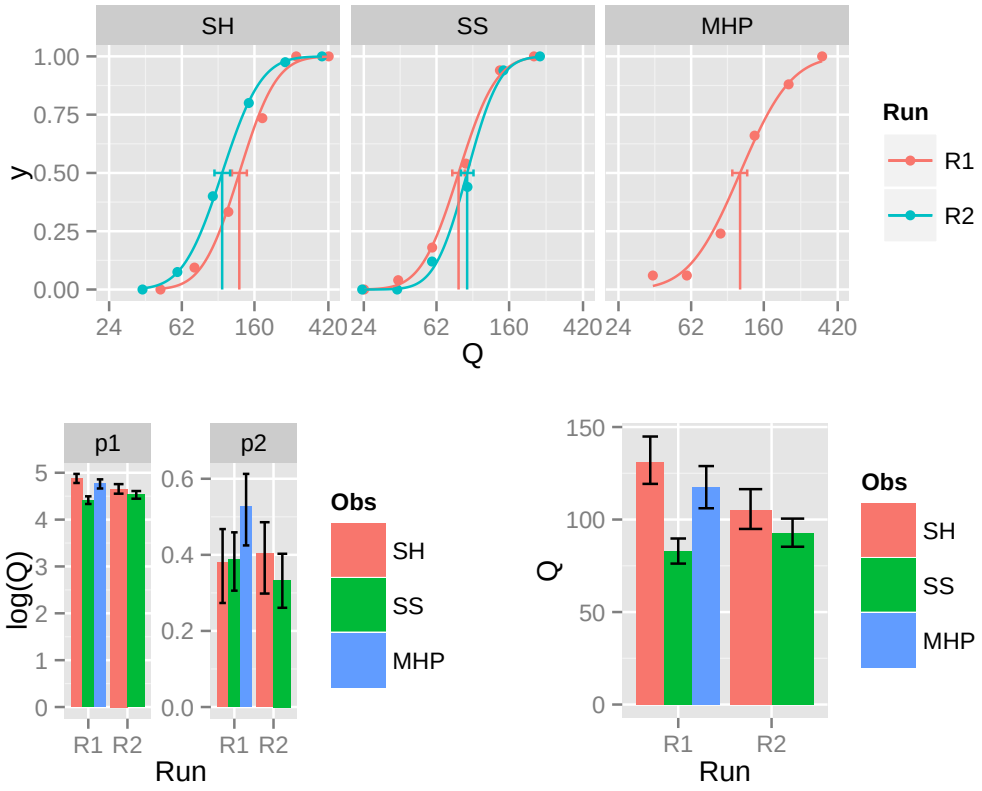


Figure 1: Psychometric functions, parameters and thresholds (including confidence intervals) resulting from fitting the model in (1) to the `HSP` data.

Knoblauch and Maloney, 2012).

To fit the model in (1) to the `HSP` dataset, we call the `quickpsy` function from package **quickpsy**

```
fit <- quickpsy(HSP, Q, k, N, grouping = .(Run, Obs), B = 1000)
```

where `B` indicates the number of bootstrap samples (which can be reduced to shorten the computation time). In this call to `quickpsy`, many arguments were not explicitly specified but left to the default values: γ and λ fixed to 0: `guess = 0` and `lapses = 0`; F set to the cumulative normal distribution: `fun = cum_normal_fun`; the probability to calculate the threshold set to 0.5: `prob = guess + 0.5 * (1 -guess)`; the confidence intervals calculated using parametric bootstrap: `bootstrap = "parametric"`.

`quickpsy` returns a list with all the information from the fitting. Most elements of the list are data frames. For example, the parameters α and β, which for the cumulative normal distribution correspond, respectively, to the mean and the standard deviation, can be found in the data frame `fit$par` (where `p1` corresponds to α and `p2` corresponds to β). The confidence intervals for the parameters are located in `fit$parci`. The thresholds and the confidence intervals for the thresholds are located in `fit$thresholds` and `fit$thresholdsci`.

`quickpsy` also returns the data frame `fit$parcomparisons`, which includes for each parameter, paired comparisons between groups for all possible pairs of groups using the bootstrap (Efron and Tibshirani, 1994). To compare two given groups, the difference between the bootstrap estimations of the parameter is calculated for all samples and from the distribution of differences, given the significance level set by the user (default: .95), the percentile confidence intervals are calculated. It is considered that the parameter differs between the two groups if the confidence intervals do not include zero. Paired comparisons for the thresholds performed using the same method are available in `fit$thresholdcomparisons`.

To plot the fitted psychometric functions, we call the **quickpsy** function `plotcurves` including the fitted model as an argument

```
plotcurves(fit)
```

To plot the parameters and the thresholds, we use functions `plotpar(fit)` and `plotthresholds(fit)` from package **quickpsy**, respectively.

Hecht et al. (1942), indeed, used $\log Q$, instead of Q as stimulus level. To easily fit and plot the

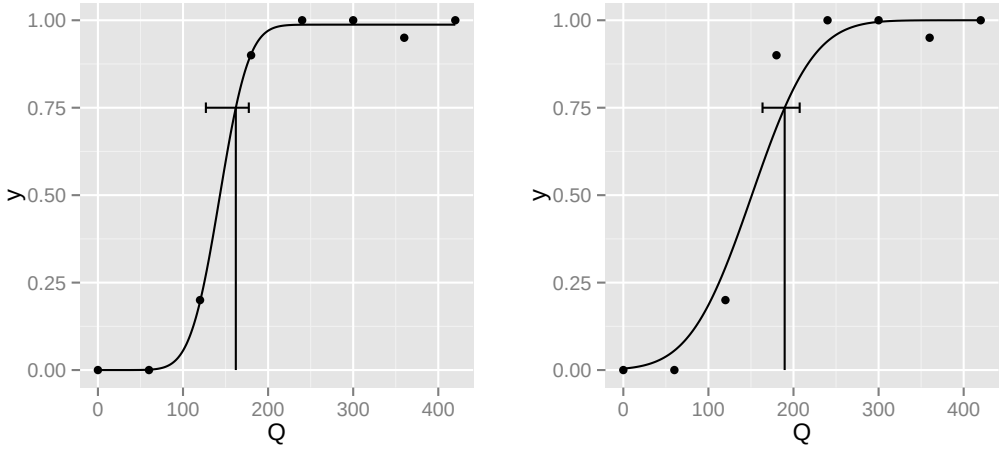


Figure 2: Psychometric function for the light detection experiment with lapses allowing λ to vary on the left and with lapses fixing λ on the right.

model using the logarithm of the stimulus level, we can call `quickpsy` with `log = TRUE`

```
fit <- quickpsy(HSP, Q, k, N, grouping = .(Run, Obs), B = 1000, log = TRUE)
```

The plots associated to the fit above (Figure 1), using package **gridExtra** (Auguie, 2015) for the plot arrangements, can be obtained as follows

```
library(gridExtra)
grid.arrange(plotcurves(fit), arrangeGrob(plotpar(fit), plotthresholds(fit), ncol = 2))
```

`HSP` is a data frame that contains summarised data: the counts of *Yes* responses. `quickpsy`, however, can fit the data frames containing more raw data in which each row corresponds to the result of one classification. In that case, the data frame should contain a response column with `1`s indicating *Yes* responses and `0`s or `-1`s indicating *No* responses and `quickpsy` should be called with the name of the response column as the `k` argument (without the argument `n` corresponding to the number of trials).

We hope that this example has illustrated that **quickpsy** requires little coding to perform typical fits and plots in a classification task with multiple groups.

Criterion-independent light detection

Following the second example in the Introduction, consider some hypothetical data in which an observer needs to decide on which from two intervals a flash of light was presented.

```
Q <- c(80, 160, 240, 320, 400) # luminance
n <- 100 # number of classifications per stimulus level
k <- c(59, 56, 69, 84, 96) # number of correct classifications
dat2IFC <- data.frame(Q, k, n)
```

To fit the model in (1) to this data, we call `quickpsy` with γ set to chance level (`guess = 0.5`)

```
fit <- quickpsy(dat2IFC, Q, k, n, guess = .5)
```

Light detection with lapses

Consider some hypothetical data from a light detection experiment, in which the observer commits a lapse (third example of the Introduction).

```
Q <- seq(0, 420, 60)
n <- 20
k <- c(0, 0, 4, 18, 20, 20, 19, 20) # lapse in the second to last intensity
datLapse <- data.frame(Q, k, n)
```

Suppose that we suspect that the observer might have committed lapses. Accordingly, we fit the model in (1) allowing λ to vary (`lapses = TRUE`).

```
fit <- quickpsy(datLapse, Q, k, n, lapses = TRUE, prob = .75)
```

The fitted psychometric function obtained with `plotcurves(fit)` is shown in Figure 2 on the left.

Typically, we are interested in the values of α, β or the threshold, which are related to the classification mechanisms (Wichmann and Hill, 2001a,b), but not in the specific value of λ. λ is allowed to vary, however, because fixing it when lapses occur can bias the estimation of α, β or the threshold (Wichmann and Hill, 2001a,b). To illustrate this, we fit the previous data with λ fixed to zero.

```
fit <- quickpsy(datLapse, Q, k, n, lapses = 0, prob = .75)
```

Figure 2 on the right shows that β and the threshold are biased.

Allowing λ to vary, however, not always *fixes* the fit. Prins (2012) has shown that, indeed, the simulations used by Wichmann and Hill (2001a,b) to illustrate how using variable λ eliminates the bias do cause a bias in threshold estimation. The following code uses **quickpsy** to replicate the results of Prins (2012). Wichmann and Hill created 7 sampling schemes for stimulus presentation that were created specifying the values of ψ, which was a Weibull function with parameters 10 and 3.

```
parweibull <- c(10, 3)
create_xs <- function(i, f) data.frame(scheme = i,  y = f,
                                       x = inv_weibull_fun(f, parweibull))
s <- list()
s[[1]] <- create_xs(1, c(.3, .4, .48, .52, .6, .7))
s[[2]] <- create_xs(2, c(.1, .3, .4, .6, .7, .9))
s[[3]] <- create_xs(3, c(.3, .44, .7, .8, .9, .98))
s[[4]] <- create_xs(4, c(.1, .2, .3, .4, .5, .6))
s[[5]] <- create_xs(5, c(.08, .18, .28, .7, .85, .99))
s[[6]] <- create_xs(6, c(.3, .4, .5, .6, .7, .99))
s[[7]] <- create_xs(7, c(.34, .44, .54, .8, .9, .98))
s <- do.call("rbind", s)
s$scheme <- factor(s$scheme)
```

Next, Wichmann and Hill simulated binomial responses using the Weibull function with several possible values for λ

```
create_sim_dat <- function(d) {
  psychometric_fun <- create_psy_fun(weibull_fun, .5, d$lambda)
  ypred <- psychometric_fun(d$x, parweibull)
  k <- rbinom(length(d$x), d$n, ypred)
  data.frame(x = d$x, k = k, n = d$n , y = k/d$n)
}
library(dplyr)
simdat <- merge(s, expand.grid(n = 160, sample = 1:100, lambda = seq(0,.05, .01))) %>%
  group_by(scheme, n, sample, lambda) %>% do(create_sim_dat(.))
```

To fit the simulated data, we use `quickpsy` bounding the possible values of λ to $[0, 0.6]$ as Wichmann and Hill did

```
fit_lapses <- quickpsy(simdat, x, k, n, within = .(scheme, lambda, sample),
                       fun = weibull_fun, bootstrap = "none", guess =.5, lapses = TRUE,
                       parini = list(c(1, 30), c(1, 10), c(0, .06)))
```

Then, we average the threshold estimation across simulations

```
thre_lapses <- fit_lapses$thresholds %>% group_by(scheme, lambda) %>%
  summarise (threshold = mean(thre))
```

and plot them including the non-biased threshold for comparison

```
real_threshold <- inv_weibull_fun((.75 - .5) / (1 - .5 - 0), parweibull)
library(ggplot2)
ggplot(thre_lapses) + geom_point(aes(x = lambda, y = threshold, color = scheme)) +
  geom_hline(yintercept = real_threshold, lty = 2)
```

Figure 3 shows, consistent with Prins (2012), that the thresholds are biased for all of the sampling schemes and that the bias increases with the λ used to simulate the data.

Appearance-based procedures

In the previous sections, we exemplified the use of **quickpsy** for *performance-based procedures* (light detection), but fitting psychometric functions is also common for *appearance-based procedures*

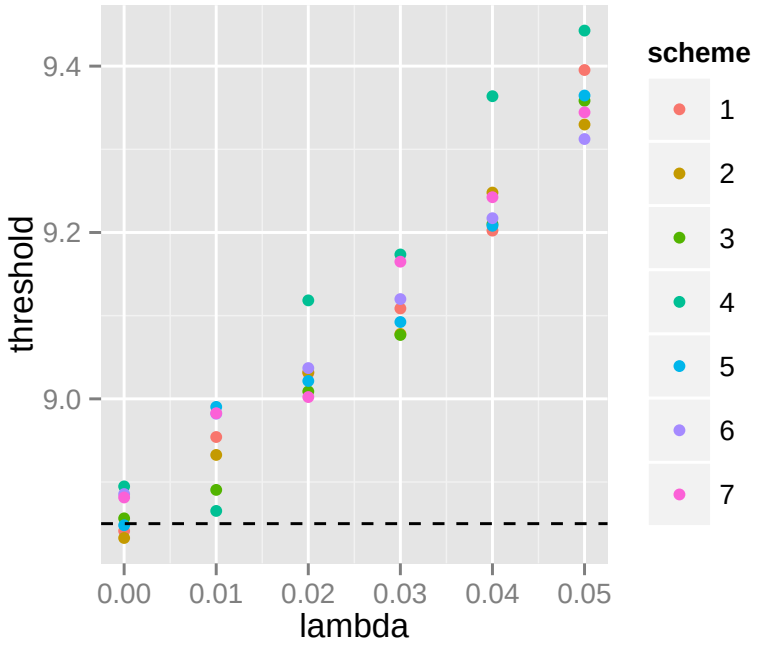


Figure 3: Threshold estimation for several sampling schemes simulating data using different values for λ. The horizontal dotted line shows the values of the unbiased threshold.

(Kingdom and Prins, 2009). For example, psychometric functions are often fitted to data measuring visual illusions such as the flash-lag (e.g., López-Moliner and Linares, 2006). In those cases, γ and λ are usually fixed to 0 and the probability summarising the behaviour of the classifier to 0.5. The stimulus level that predicts 0.5 proportion of *Yes* responses is called the *point of subjective equality (PSE)*.

Implementation details

Non-standard evaluation and grouping

`quickpsy`, the main function of **quickpsy**, is a *non-standard evaluation NSE* function and, as such, the names of the arguments and not only their values can be accessed (Wickham, 2014). NSE functions, which are common in R, are useful for example to label the axes of a plot using the name of the arguments (Wickham, 2014). As calling NSE functions from other functions is difficult (Wickham, 2014), **quickpsy** also incorporates `quickspsy_` which is the *standard evaluation SE* version of `quickpsy`. To call SE functions, some of the arguments need to be quoted. The following code exemplifies the use of `quickspsy_` to fit the `HSP` dataset from the first example of usage

```
fit <- quickpsy_(HSP, "Q", "k", "N", grouping = c("Run", "Obs"))
```

The NSE high-level plotting functions of **quickpsy** `plotcurves`, `plotpar` and `plotthresholds` also have associated SE versions: `plotcurves_`, `plotpar_` and `plotthresholds_`.

One of the main features of **quickpsy** is the possibility of fitting multiple groups. To handle the data analysis and plotting for multiple groups, **quickpsy** relies on **dplyr** (Wickham and Francois, 2015) and **ggplot2** (Wickham, 2009) respectively. In particular, **quickpsy** makes extensive use of the function `do` from **dplyr**, which splits input data frames by group, applies a function to each group and returns an output data frame. `quickpsy`, for example, calls the functions `curves` and `thresholds`, which basically contain a `do` function that, in turn, calls `one_curve` and `one_threshold`, which are the functions that calculate the psychometric curve and the threshold for a group of data (this method of function `X` calling function `one_X`, which performs the computations for a group of data, is used for some other functions called from `quickpsy`).

Closures

Closures or *function factories* are functions written by functions (Wickham, 2014). When estimating the maximum likelihood parameters for the model in (1), two procedures can be executed naturally

using closures. One is the creation of ψ from F (and the parameters γ and λ), which is implemented in the `create_psy_fun` closure. The other is the creation of the negative log likelihood function from ψ (and the data), which is implemented in the `create_nll` closure.

Round-off errors in the log likelihood

Consider that we want to *manually* fit a cumulative normal psychometric function to the following data

```
x <- c(-0.056, 0.137, 0.331, 0.525, 0.719, 0.912, 1.100)
k <- c(0, 5, 11, 12, 12, 12, 12)
n <- c(12, 12, 12, 12, 12, 12, 12)
```

by direct minimisation of the negative log likelihood. In **quickpsy** we build the negative log likelihood using a function similar to

```
nll <- function(p) {
  phi <- pnorm(x, p[1], p[2])
  -sum(k * log(phi) + (n - k) * log(1 - phi))
}
```

and use `optim` to find the parameters that minimise it

```
optim(c(0.5, 0.1), nll)
```

`optim` requires initial values for the parameters that we arbitrarily chose as `c(0.5,0.1)`. But suppose that we choose another set of initial parameters

```
optim(c(0.1, 0.1), nll)
```

This time `optim` returns an error:

```
Error in optim(c(0.1, 0.1), nll) :
  function cannot be evaluated at initial parameters
```

The problem is that for the `1.100` stimulus level, `pnorm(1.100,0.1,0.1)` is rounded to `1` and therefore the term `log(1 - phi)` in the negative log likelihood is `-Inf`.

To avoid this problem, the coding of the negative log likelihood in **quickpsy** incorporates the following lines

```
 phi[phi < .Machine$double.eps] <- .Machine$double.eps
 phi[phi > (1 - .Machine$double.eps)] <- 1 - .Machine$double.eps
```

That is, values that are smaller than the machine accuracy (`.Machine$double.eps`) are replaced by `.Machine$double.eps` and values that are larger than `1 - .Machine$double.eps` are replaced by `1 - .Machine$double.eps`. We can verify that `quickpsy` does not produce errors when using the *problematic* initial values with the following code

```
dat <- data.frame(x, k, n)
fit <- quickpsy(dat, x, k, n, parini = c(0.1, 0.1))
```

quickpsy allows to use the cumulative normal function, but also the cumulative logistic, cumulative Weibull or any other cumulative distribution function defined by the user for modeling the probability of responding *Yes*. The function `nll` evaluating the log likelihood was thus defined in **quickpsy** similar to the way above in order to encompass all these variants. Note, however, that if the aim was to fit only a cumulative normal function, the negative log likelihood function could be defined in the following way in R to cover a larger range of possible values of `p` without problems by directly evaluating the cumulative distribution function on the log scale

```
nll <- function(p) {
  logPhi <- pnorm(x, p[1], p[2], log.p = TRUE)
  log1mPhi <- pnorm(x, p[1], p[2], lower.tail = FALSE, log.p = TRUE)
  -sum(k * logPhi + (n - k) * log1mPhi)
}
```

Optimisation and initial parameters

By default, `quickpsy` searches the minimum of the negative log likelihood using `optim`, which requires initial values for the parameters. To free the user from the necessity of providing initial parameters,

`quickpsy` uses as initial values the parameters obtained by linear modelling of the probit-transformed data (McKee et al., 1985; Gescheider, 1997). Linear modelling probit-transformed data is a poor technique to estimate the parameters of the psychometric function (McKee et al., 1985), but our tests suggest that they are good enough to be used as initial parameters.

The user can overwrite the initial parameters calculated using probit-transformed data by providing a vector of initial parameters to the argument `parini` of `quickpsy`. A list of vectors can also be fed when the user wants to set up some bounds for the parameters (see the last example of usage).

Acknowledgments

We thank Kenneth Knoblauch for providing helpful comments on a draft version of the manuscript and helping us with the problem associated to the rounding-off errors in the calculation of the likelihood. The research group is supported by Grant 2014SGR79 from the Catalan government. J. López-Moliner was supported by an ICREA Academic Distinguished Professorship award and grant PSI2013-41568-P from MINECO.

Bibliography

B. Auguie. *gridExtra: Miscellaneous Functions for 'grid' Graphics*, 2015. URL http://CRAN.R-project.org/package=gridExtra. R package version 2.0.0. [p126]

B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. CRC Press, 1994. [p125]

J. Fischer and D. Whitney. Serial dependence in visual perception. *Nature Neuroscience*, 17(5):738–743, May 2014. [p122]

I. Fründ, N. V. Haenel, and F. A. Wichmann. Inference for psychometric functions in the presence of nonstationary behavior. *Journal of Vision*, 11(6), 2011. [p122, 124]

G. A. Gescheider. *Psychophysics*. The Fundamentals. Psychology Press, June 1997. [p123, 124, 130]

J. I. Gold and L. Ding. How mechanisms of perceptual decision-making affect the psychometric function. *Progress in Neurobiology*, 103:98–114, Apr. 2013. [p122]

J. I. Gold and M. N. Shadlen. The neural basis of decision making. *Annual Review of Neuroscience*, 30: 535–574, 2007. [p122]

D. M. Green and J. A. Swets. *Signal Detection Theory and Psychophysics*. Wiley, 1966. [p123, 124]

S. Hecht, S. Shlaer, and M. H. Pirenne. Energy, quanta, and vision. *Journal of General Physiology*, 25(6): 819–840, July 1942. [p124, 125]

F. A. A. Kingdom and N. Prins. *Psychophysics*. A Practical Introduction. Academic Press, Sept. 2009. [p122, 123, 124, 128]

S. A. Klein. Measuring, estimating, and understanding the psychometric function: A commentary. *Perception and Psychophysics*, 63(8):1421–1455, Nov. 2001. [p122]

K. Knoblauch. *psyphy: Functions for Analyzing Psychophysical Data in R*, 2014. URL http://CRAN.R-project.org/package=psyphy. R package version 0.1-9. [p124]

K. Knoblauch and L. T. Maloney. *Modeling Psychophysical Data in R*. Springer, New York, NY, Sept. 2012. [p122, 123, 124, 125]

M. Kuss, F. Jäkel, and F. A. Wichmann. Bayesian inference for psychometric functions. *Journal of Vision*, 5(5):478–492, 2005. [p123]

D. Linares and J. López-Moliner. *quickpsy: Fits Psychometric Functions for Multiple Groups*, 2016. URL https://CRAN.R-project.org/package=quickpsy. R package version 0.1.3. [p124]

J. López-Moliner and D. Linares. The flash-lag effect is reduced when the flash is perceived as a sensory consequence of our action. *Vision Research*, 46(13):2122–2129, June 2006. [p128]

Z.-L. Lu and B. Dosher. *Visual Psychophysics*. From Laboratory to Theory. MIT Press, Oct. 2013. [p122, 123, 124]

N. A. Macmillan and C. D. Creelman. *Detection Theory*. A User's Guide. Psychology Press, Sept. 2004. [p122, 123]

I. Marin-Franch, K. Zychaluk, and D. H. Foster. *modelfree: Model-Free Estimation of a Psychometric Function*, 2012. URL http://CRAN.R-project.org/package=modelfree. R package version 1.1-1. [p124]

S. P. McKee, S. A. Klein, and D. Y. Teller. Statistical properties of forced-choice psychometric functions: Implications of probit analysis. *Perception and Psychophysics*, 37(4):286–298, 1985. [p130]

A. Moscatelli, M. Mezzetti, and F. Lacquaniti. Modeling psychophysical data at the population-level: The generalized linear mixed model. *Journal of Vision*, 12(11), 2012. [p124]

J. C. Nash. *Nonlinear Parameter Optimization Using R Tools*. John Wiley & Sons, New York, 2014. [p123]

J. K. O'Regan and R. Humbert. Estimating psychometric functions in forced-choice situations: Significant biases found in threshold and slope estimations when small samples are used. *Perception and Psychophysics*, 46(5):434–442, Nov. 1989. [p122, 123]

N. Prins. The psychometric function: The lapse rate revisited. *Journal of Vision*, 12(6), 2012. [p127]

R. F. Quick. A vector-magnitude model of contrast detection. *Kybernetik*, 16(2):65–67, 1974. [p124]

C. Summerfield and K. Tsetsos. Do humans make good decisions? *Trends in Cognitive Sciences*, 19(1): 27–34, Jan. 2015. [p122]

E. Van der Burg, D. Alais, and J. Cass. Rapid recalibration to audiovisual asynchrony. *Journal of Neuroscience*, 33(37):14633–14637, Sept. 2013. [p122]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Statistics and Computing. Springer, New York, NY, 2002. [p124]

L. Wasserman. *All of Statistics*. A Concise Course in Statistical Inference. Springer, New York, NY, Dec. 2013. [p123]

A. B. Watson. Probability summation over time. *Vision Research*, 19(5):515–522, 1979. [p122, 123]

A. B. Watson and J. A. Solomon. Psychophysica: Mathematica notebooks for psychophysical experiments. *Spatial Vision*, 10(4):447–466, 1997. [p124]

F. A. Wichmann and N. J. Hill. The psychometric function: I. Fitting, sampling, and goodness of fit. *Perception and Psychophysics*, 63(8):1293–1313, Nov. 2001a. [p122, 123, 127]

F. A. Wichmann and N. J. Hill. The psychometric function: II. Bootstrap-based confidence intervals and sampling. *Perception and Psychophysics*, 63(8):1314–1329, Nov. 2001b. [p123, 127]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer, New York, 2009. URL http://had.co.nz/ggplot2/book. [p128]

H. Wickham. *Advanced R*. CRC Press, Sept. 2014. [p124, 128]

H. Wickham and R. Francois. *dplyr: A Grammar of Data Manipulation*, 2015. URL http://CRAN.R-project.org/package=dplyr. R package version 0.4.3. [p128]

K. Zychaluk and D. H. Foster. Model-free estimation of the psychometric function. *Attention, Perception & Psychophysics*, 71(6):1414–1425, Aug. 2009. [p124]

Daniel Linares
Vision and Control of Action Group, Departament de Psicologia Bàsica, Universitat de Barcelona
Passeig Vall d' Hebron 171, Barcelona 08035
Spain
danilinares@gmail.com

Joan López-Moliner
Vision and Control of Action Group, Departament de Psicologia Bàsica, Universitat de Barcelona
Institute for Brain, Cognition and Behaviour (IR3C), Barcelona
Passeig Vall d' Hebron 171, Barcelona 08035
Spain
j.lopezmoliner@ub.edu

FWDselect: An R Package for Variable Selection in Regression Models

by Marta Sestelo, Nora M. Villanueva, Luis Meira-Machado and Javier Roca-Pardiñas

Abstract In multiple regression models, when there are a large number (p) of explanatory variables which may or may not be relevant for predicting the response, it is useful to be able to reduce the model. To this end, it is necessary to determine the best subset of q ($q \leq p$) predictors which will establish the model with the best prediction capacity. **FWDselect** package introduces a new forward stepwise-based selection procedure to select the best model in different regression frameworks (parametric or nonparametric). The developed methodology, which can be equally applied to linear models, generalized linear models or generalized additive models, aims to introduce solutions to the following two topics: i) selection of the best combination of q variables by using a step-by-step method; and, perhaps, most importantly, ii) search for the number of covariates to be included in the model based on bootstrap resampling techniques. The software is illustrated using real and simulated data.

Introduction

In a multivariate regression framework, the target response Y can depend on a set of p initial covariates $X_1, X_2, \ldots, X_p$ but in practical situations we often would like to determine which covariates are "relevant" to describe this response.

The question of how to choose a subset of predictors of size q ($q \leq p$) has not totally been satisfactorily solved yet. This problem is particularly important for large p and/or when there are redundant predictors. As a general rule, an increase in the number of variables to be included in a model provides an "apparently" better fit of the observed data; however, these estimates are not always satisfying for different reasons. On the one hand, the inclusion of such irrelevant variables would increase the variance of the estimates, resulting in a partial loss of the predictive capability of the model. On the other hand, the inclusion of too many variables may lead to unnecessary complexity in the resulting model, conducing to a difficult interpretation.

Model selection (and variable selection in regression, in particular) is a trade-off between bias and variance. Inference based on models with few variables can be biased, however, models that take into account too many variables may result in a lack of precision or false effects. These considerations call for a balance between under- and over-fitted models, the so-called model-selection problem (Forster, 2000).

To solve this problem, several strategies have been proposed. One common option is to use iterative procedures, such as the leaps and bounds algorithm (Furnival and Wilson, 1974) through which the best subset selection is obtained. This is a full information criteria-based approach, which compares all possible models and ranks them (Calcagno and de Mazancourt, 2010). Nevertheless, the problem of selecting the best model from among all possible combinations of p predictors is not trivial. In the presence of a large number of variables this selection procedure may require an excessively high computational cost and thus, in some cases, the problem becomes intractable. In order to relax this exhaustive search, heuristic iterative procedures such as forward- and backward-stepwise (Hocking, 1976) have been developed. This greedy algorithm produces a nested sequence of models based on the use of some information criteria which compares the models obtained in the course of the simplification or complexification scheme. Several criteria have been used for this purpose (Venables and Ripley, 1997; Miller, 2002), including Mallow's Cp (Mallows, 1973) or the Akaike Information Criteria or AIC (Akaike, 1973). Apart from the iterative procedures, other strategies applied in the variable selection problem are, e.g. shrinkage regression methods —such as ridge regression or the Lasso (least absolute shrinkage and selection operator) (Tibshirani, 1996; Hastie et al., 2003)— or the Bayesian approach (Green, 1995; Kuo and Mallick, 1998; Park and Casella, 2008; Hans, 2009).

Several R packages have been developed to carry out automatic variable selection or model selection. For instance, the **meifly** package (Wickham, 2014) can be used to search through all the different models. This type of exhaustive search can be also addressed using some other algorithm, such as the branch-and-bound algorithm in the **leaps** package (Lumley and Miller, 2009) or the leaps-and-bounds algorithm in the **subselect** package (Orestes Cerdeira et al., 2015). Both packages also implement other selection methods (heuristics). The **leaps** package includes the forward or backward stepwise, or sequential replacement while the **subselect** package provides a simulated annealing-type search algorithm, a genetic algorithm, and a restricted local improvement algorithm. To use the Lasso method, the user can apply, for example, the `lars` function implemented in the **lars** package (Hastie and Efron, 2013) or the `glmnet` function, which fits a generalized linear model

via penalized maximum likelihood, implemented in the **glmnet** package (Friedman et al., 2015). Additionally, another procedure used by the R community seems to be the model-selection-oriented `step` function (Hastie and Pregibon, 1992) built into the **stats** package. When it comes to model selection with generalized linear models, one option could be to use the **glmulti** package (Calcagno, 2013) or **bestglm** package (Mcleod and Xu, 2014). Finally, within the class of generalized additive models, other algorithms have also been introduced to achieve component selection, see Lin and Zhang (2006) and references therein, the boosting technique of Tutz and Binder (2006) or the generalization of the approach of Belitz and Lang (2008). More recently, and widely applied by R users, the `gam` function of the **mgcv** package (Wood, 2006, 2011) includes an argument (`select = TRUE`) for model selection and fitting in a single step by adding a second penalty term in the estimation scheme of continuous covariates (Marra and Wood, 2011).

The **FWDselect** package introduces an alternative to existing approaches in the form of a methodology whereby R users can select the best variables in different regression contexts, both parametric and nonparametric. Unlike other existing packages, the procedure implemented in it can be equally applied to linear models, generalized linear models or/and generalized additive models, with Gaussian, binary or Poisson response. The forward selection algorithm proposed is based on a greedy procedure which changes one variable at the time in the model — keeping the others fixed — and does this repeatedly until none of the selected variables can be exchanged to improve model fit. This is a greedy algorithm, which may not find the actual best solution, but is less greedy than other methods such as step. In addition, in contrast with other packages in which the users must decide — either previous or post selection — the number of variables that have to be included, the bootstrap test introduced in our procedure allows them to determine the number with a significance level.

The remainder of this paper is organised as follows. First we describe the algorithm used to select the best subset of size q, along with the bootstrap techniques that are used to determine the number of variables to be included in the model. Then a detailed description of the package is presented, and its usage is illustrated through the analysis of three data sets. Finally, the last section contains the main conclusions of this work.

Methodology background

This section introduces the developed methodology and gives a description of the variable selection algorithm. The implemented package can be used with Gaussian, binary or Poisson response, however and based on the application data, we will explain the procedure with a nonparametric regression model with Gaussian response.

Let $\mathbf{X} = (X_1, X_2, \ldots, X_p)$ be a vector of p initial variables and Y the response. An additive regression model can be expressed as

$$Y = m(\mathbf{X}) + \varepsilon, \tag{1}$$

where

$$m(\mathbf{X}) = \alpha + m_1(X_1) + m_2(X_2) + \cdots + m_p(X_p),$$

where $m_j (j = 1, \ldots, p)$ are smooth and unknown functions and ε is the zero-mean error. Additionally, to guarantee the identification of the above model, a constant α is introduced in the model and it is required that the partial functions satisfy

$$E\left[m_j\left(X_j\right)\right] = 0, \quad j = 1, \ldots, p.$$

This implies that $E[Y] = \alpha$.

To date, several approaches to estimating the model in (1) have been suggested in the statistical literature, e.g., Buja et al. (1989), Härdle and Hall (1993), Mammen et al. (1999). In this package penalized regression splines, as implemented in the **mgcv** package, are used.

It is important to highlight that, in situations involving a large number of variables, correct estimation of the response will be obtained on the basis of selecting the appropriate predictors. In the case that we have information a priori about which of the initial set of variables are relevant, it would be possible to apply a likelihood ratio test (Neyman and Pearson, 1928) or a F-test type (Seber and Wild, 1989; Seber, 1997) in a parametric framework, or a generalized likelihood ratio test (Fan et al., 2001; Fan and Jiang, 2005, 2007) in a nonparametric one. However, in situations where we do not have information in advance, it will be necessary to select the model according to a selection algorithm.

According to this, we propose a procedure that includes two topics: i) selecting the best combination of q variables by using a new forward stepwise-based selection procedure; and ii) determining the minimum number of covariates to be included in the model. Both topics are explained as below.

Selecting the best variables

The first topic of our procedure is, given a number q $(q \leq p)$, to select the best combination of q variables. For this purpose, one option is to use a complete subset selection method as Roca-Pardiñas et al. (2009), which requires all possible models to be considered. When confronted with a large number of variables, however, the computational cost of the procedure can be very high or even prohibitive. In view of this, we provide a new method that speeds up the process based on a heuristic search which aims to approximate the optimal solution. There is no guarantee however that the procedure finds the best subset of covariates — this could only be achieved based on searching through all the possible subsets — but it has the advantage of requiring a smaller number of computations to reach the optimal solution or, at least, close to the optimal one.

Let $X_{j_1}, \ldots, X_{j_k}$ be a subset of variables of size k $(k \leq q)$. We define $IC_{j_1,\ldots,j_k}$ as one possible information criterion (such as AIC, deviance, residual variance, etc.) of the nonparametric model

$$Y = \alpha + m_{j_1}\left(X_{j_1}\right) + m_{j_2}\left(X_{j_2}\right) + \ldots + m_{j_k}\left(X_{j_k}\right) + \varepsilon', \tag{2}$$

where ε' is the zero-mean error. Based on this information criterion, IC, the proposed automatic forward stepwise selection method is given in Algorithm 1. Note that any criterion can be used without correcting it to take account of the number of variables. This is possible because the models which are compared have always the same number of variables.

Testing the number of significant variables

Previously, the best subset of q variables is selected according to an information criterion. However, the question that arises in this procedure is to know the optimal number q. Thus, the second topic in our methodology is to decide the number of covariates that should be included in the model, i.e, determining the number of significant variables.

Accordingly, we propose a procedure to test the null hypothesis of q significant variables in the model versus the alternative in which the model contains more than q variables. Based on the additive model

$$Y = m(\mathbf{X}) + \varepsilon \quad \text{where} \quad m(\mathbf{X}) = \alpha + m_1(X_1) + m_2(X_2) + \cdots + m_p(X_p),$$

the following strategy is considered: for a subset of size q, considerations will be given to test the null hypothesis

$$H_0(q) : \sum_{j=1}^{p} I_{\{m_j \neq 0\}} \leq q$$

versus the general hypothesis

$$H_1(q) : \sum_{j=1}^{p} I_{\{m_j \neq 0\}} > q,$$

where I is the indicator function and considering that the m_j's are not equal to zero on a set of positive probability.

Given a i.i.d. sample $\{(\mathbf{X}_i, Y_i)\}_{i=1}^{n}$, with $\mathbf{X} = (X_1, \ldots, X_p)$, to test the above null hypothesis we propose the following strategy:

(i) Obtain the best subset of q predictor variables. To this end we use the selection algorithm described in Algorithm 1. Without loss of generality, we assume that the q variables selected are in the first q positions of the $\mathbf{X}$ vector.

(ii) Obtain the nonparametric estimates of the null model as

$$\hat{m}_0(\mathbf{X}_i) = \alpha + \hat{m}_1(X_{i1}) + \ldots + \hat{m}_q\left(X_{iq}\right).$$

(iii) Compute the residuals as $r_i = Y_i - \hat{m}_0(\mathbf{X}_i)$ and obtain the nonparametric estimates of $g(\mathbf{X}_i)$ according to the model

$$r_i = g(\mathbf{X}_i) + \varepsilon, \tag{4}$$

Algorithm 1: Modified forward stepwise selection method

1. Given a number q, selects the q variables $X_{l_1}, \ldots, X_{l_q}$ which minimises the following expression

$$(l_1, l_2, \ldots, l_q) = \underset{\substack{(j_1,\ldots,j_q) \\ 1\leq j_1\leq\cdots\leq j_q\leq p}}{\arg\min} \; IC_{j_1,\ldots,j_q}. \tag{3}$$

2. The elements of the vector of indices $(l_1, l_2, \ldots, l_q)$ are selected consecutively in the following manner:

   (a) Firstly, determine the variable of the first position X_{l_1} where

   $$l_1 = \underset{\substack{j_1 \\ 1\leq j_1\leq p}}{\arg\min} \; IC_{j_1}.$$

   Note that all possible models of one variable must be estimated.

   (b) Fix the first variable obtained previously, X_{l_1} , and obtain the second one, X_{l_2}, with

   $$l_2 = \underset{\substack{j_2 \\ 1\leq j_2\leq p,\; j_2\neq l_1}}{\arg\min} \; IC_{l_1,j_2}.$$

   (c) Fix X_{l_1} and X_{l_2}, and obtain the third one, X_{l_3}, where

   $$l_3 = \underset{\substack{j_3 \\ 1\leq j_3\leq p,\; j_3\notin\{l_1,l_2\}}}{\arg\min} \; IC_{l_1,l_2,j_3}.$$

   (d) Fix $X_{l_1}, X_{l_2}, \ldots, X_{l_{q-1}}$, and repeat the procedure analogously until the q-th variable, X_{l_q}, with

   $$l_q = \underset{\substack{j_q \\ 1\leq j_q\leq p,\; j_q\notin\{l_1,\ldots,l_{q-1}\}}}{\arg\min} \; IC_{l_1,\ldots,j_q}$$

3. Once variables $X_{l_1}, X_{l_2}, \ldots, X_{l_q}$ have been selected, run through positions $j = 1, \ldots, q$ and replace each l_j element as follows, only if the obtained IC is less than the minimum criterion obtained with the previous l_j,

$$l_j = \underset{\substack{j_j \\ j_j\notin\{l_1,\ldots,l_{j-1},l_{j+1},\ldots,l_q\}}}{\arg\min} \; IC_{l_1,\ldots,l_{j-1},j_j,l_{j+1},\ldots,l_q}.$$

4. Step 3 is repeated until there is no change in the selected covariates, i.e., the algorithm stops when it has gone through a complete cycle without changing any of the q positions.

where g is an unknown and smooth function which is applied to a unique covariate. This covariate will be chosen from $X_{q+1}, \ldots, X_p$ applying the selection algorithm exposed in Algorithm 1. Without loss of generality, we assume that $g(\mathbf{X}) = \alpha + g_{q+1}(X_{q+1})$.

The purpose of this step is to assess whether there is enough structure left in the residuals that could be modeled by the predictors not included in the null model. Note that, within these possible predictors, we only select one of them in order to reduce the computational cost of the algorithm. However, the ideal solution would be to estimate the model in (4) determining the best subset of predictors within the remainder variables, instead of selecting only one of them.

Both options are implemented in the package by means of the `speedup` argument of the `test` function. If `speedup = TRUE` is specified a unique predictor for the residuals is used. If `speedup = FALSE` is specified the user can choose more than one predictor. With this latter option, when the number of variables is large, the selection of the best subset of predictors for the residuals requires a high computational burden[1]. Therefore, in practice, we propose a solution by using the `qmin` argument which must be filled by the user. This argument corresponds to the size of the best subset of predictors. In order to help the user select it, it is recommended to visualize the graphical output of the plot function and to choose the number q which minimizes the curve.

(iv) Finally, we propose the following test statistic, based on the estimations of g

$$T = \sum_{i=1}^{n} |\hat{g}(\mathbf{X}_i)|.$$

It is important to stress that, if the null hypothesis holds, T should be close to zero. Thus, the test rule for checking $H_0(q)$ with a significance level of α is that the null hypothesis is rejected if T is larger than its $(1-\alpha)$-percentile. To approximate the distributions of the test statistic resampling methods such as the bootstrap introduced by Efron (1979) (see also Efron and Tibshirani, 1993; Härdle and Mammen, 1993; Kauermann and Opsomer, 2003) can be applied. Here we use the wild bootstrap (Wu, 1986; Liu, 1988; Mammen, 1993) because this method is valid both for homocedastic and heteroscedastic models where the variance of the error is a function of the covariate. The testing procedure consists of the following steps:

Step 1: Obtain T from the sample data, as explained above.

Step 2: For $i = 1, \ldots, n$, obtain $\hat{m}_0(\mathbf{X}_i)$ and the bootstrap residuals as

$$\varepsilon_i^{\bullet b} = \hat{\varepsilon}_i V_i$$

where $\hat{\varepsilon}_i = Y_i - \hat{m}_0(\mathbf{X}_i)$ are the residuals of the null model and $V_1, \ldots, V_n$ is an i.i.d. random variable with mass $(5+\sqrt{5})/10$ and $(5-\sqrt{5})/10$ at the points $(1-\sqrt{5})/2$ and $(1+\sqrt{5})/2$. Note that this distribution satisfies $E(V_i) = 0, E(V_i^2) = E(V_i^3) = 1$.

Step 3: For $b = 1, \ldots, B$, simulate the bootstrap sample $\left\{\mathbf{X}_i, Y_i^{\bullet b}\right\}_{i=1}^{n}$ with $Y_i^{\bullet b} = \hat{m}_0(\mathbf{X}_i) + \varepsilon_i^{\bullet b}$, and compute the bootstrap estimates of $T^{\bullet b}$.

The test rule based on T is given by rejecting the null hypothesis if $T > T^{1-\alpha}$, where $T^{1-\alpha}$ is the empirical $(1-\alpha)$-percentile of values $T^{\bullet b}$ $(b = 1, \ldots, B)$.

Applying this test to $q = 1, \ldots, p-1$ could be an important issue in a covariate selection procedure. If $H_0(q)$ is not rejected, only the subset of the covariates $X_{j_1}, \ldots, X_{j_q}$ will be retained, and the remaining variables will be eliminated from the model. In all other cases, the test is repeated with $q+1$ variables until the null hypothesis is not rejected. For example, if $H_0(1)$ is not rejected just one variable should be included into the model. If this hypothesis is rejected it will be required to test $H_0(2)$. If this new hypothesis is again rejected, $H_0(3)$ should be tested and so on until a certain $H_0(q)$ is accepted.

The validation of the approach relying on the bootstrap-based test can be consulted in Sestelo (2013) where type I error and power have been calculated for different test statistics. Also, the performance of the test for different levels of correlation between covariates have been analyzed. All the test statistics perform reasonably well, with the level coming relatively close to the nominal size and the probability of rejection rising as we separate from the null hypothesis, specially with large sample sizes. Furthermore, several simulation studies have been considered in order to compare the methodology proposed in this paper with other procedures reported in the literature that carry out automatic variable selection.

[1]The procedure for selecting the best subset of predictors for the residuals would be as follows: for each possible v value ($v = 1, \ldots, p-q$), it should be used the Algorithm 1 to identify the best v variables and to obtain the IC_v from the fitted model with them. Then, it should be looked at all of the resulting models, with the goal of identifying the one that is best, i.e., the model with the minimum IC_v.

FWDselect in practice

This section introduces an overview of how the package is structured. **FWDselect** is a shortcut for "Forward selection" and this is its major functionality: to provide a forward stepwise-based selection procedure. This software helps the user select relevant variables and evaluate how many of these need to be included in a regression model. In addition, it enables both numerical and graphical outputs to be displayed.

Our package includes several functions that enable users to select the variables to be included in linear models, generalized linear models or generalized additive models. The functions within **FWDselect** are briefly described in Table 1.

Users can obtain the best combinations of q variables by means of the main function which is `selection`. Additionally, if one wants to obtain the results for more than one subset size, it is possible to apply the `qselection` function, which returns a summary table showing the different subsets, selected variables and information criterion values. These values are obtained by cross-validation with the purpose of comparing correctly the resulting models which include a different number of variables. The object obtained with this last function is the argument required for `plot`, which provides a graphical output. Finally, to determine the number of variables that should be introduced in the model, only the `test` function needs to be applied. Table 2 provides a summary of the arguments of the `selection`, `qselection` and `test` functions. The most computationally demanding parts of the code, namely those that involve the estimation of the models, the cross-validation and the bootstrap, have been parallelized by means of the **parallel** package via forking on Unix-alike platforms or creating a PSOCK cluster on Windows systems.

Function	Description
`selection`	Main function for selecting a subset of q variables. Note that the selection procedure can be used with `lm`, `glm` or `gam` functions.
`print.selection`	Method of the generic print function for `"selection"` objects, which returns a short summary.
`qselection`	Function that enables users to obtain the selected variables for more than one size of subset.
`print.qselection`	Method of the generic print function for `"qselection"` objects. Returns a table showing the chosen covariates to be introduced into the models and their information criteria obtained by cross-validation.
`plot.qselection`	Visualisation of `"qselection"` objects. Plots the cross-validation information criterion for several subsets with size q chosen by users.
`test`	Function that applies a bootstrap-based test for covariate selection. Helps determine the precise number of variables to be included in the model.

Table 1: Summary of functions in the **FWDselect** package.

Example of application

In this section we illustrate the use of **FWDselect** package using a real data set, the pollution data (included in the package). The software is applied to the prediction of atmospheric SO_2 pollution incidents by means of additive models. Combustion of fuel oil or coal releases sulphur dioxide into the atmosphere in different quantities. Current Spanish legislation governing environmetrical pollution controls the vicinity of potential point sources of pollution, such as coal-fired power stations. It places a limit on the mean of 24 successive determinations of SO_2 concentration taken at 5-minute intervals. An emission episode is said to occur when the series of bi-hourly means of SO_2 is greater than a specific level, r. In this framework, it is of interest for a plant, both economically and environmentally, to be able to predict, when the legal limit will be exceeded with sufficient time for effective countermeasures to be taken.

In previous studies (García-Jurado et al.; Prada-Sánchez et al., 2000; Prada-Sánchez and Febrero-Bande, 1997; Roca-Pardiñas et al., 2004), semiparametric, partially linear models and generalized additive models with unknown link functions were applied to the prediction of atmospheric SO_2 pollution incidents in the vicinity of a coal/oil-fired power station. Here, we present a new approach to this problem, whereby we try to predict a new emission episode, focusing our attention on the importance of ascertaining the best combinations of time instants for the purpose of obtaining the best prediction. Bearing this in mind, the selection of the optimal subset of variables could be a good approach to this issue.

	`selection()` arguments
`x`	A data frame containing all the covariates.
`y`	A vector with the response values.
`q`	An integer specifying the size of the subset of variables to be selected.
`prevar`	A vector containing the number of the $q-1$ selected variables in the previous step. By default it is NULL.
`criterion`	The information criterion to be used. Default is the `"deviance"`. Other functions provided are the coefficient of determination (`"R2"`), the residual variance (`"variance"`), the Akaike information criterion (`"aic"`), AIC with a correction for finite sample sizes (`"aicc"`) and the Bayesian information criterion (`"bic"`). The deviance, coefficient of determination and variance are calculated by cross-validation.
`method`	A character string specifying which regression method is used, `"lm"`, `"glm"` or `"gam"`.
`family`	This is a family object specifying the distribution and link to use in fitting.
`seconds`	A logical value. If `TRUE` then, rather than returning the single best model only, the function returns a few of the best models.
`nmodels`	Number of secondary models to be returned.
`nfolds`	Number of folds for the cross-validation procedure, for `deviance`, `R2` or `variance` criterion.
`cluster`	A logical value. If `TRUE` (default) the code is parallelized. Note that there are cases without enough repetitions (e.g., a low number of initial variables) that R will gain in performance through serial computation. R takes time to distribute tasks across the processors also it will need time for binding them all together later on. Therefore, if the time for distributing and gathering pieces together is greater than the time needed for single-thread computing, it could be better not to parallelize.
`ncores`	An integer value specifying the number of cores to be used in the parallelized procedure. If `NULL`, the number of cores to be used is equal to the number of cores of the machine -1.
	`qselection()` arguments
`x`	A data frame containing all the covariates.
`y`	A vector with the response values.
`qvector`	A vector with more than one variable-subset size to be selected.
`criterion`	The information criterion to be used. Default is the `"deviance"`. Other functions provided are the coefficient of determination (`"R2"`), the residual variance (`"variance"`), the Akaike information criterion (`"aic"`), AIC with a correction for finite sample sizes (`"aicc"`) and the Bayesian information criterion (`"bic"`). The deviance, coefficient of determination and variance are calculated by cross-validation.
`method`	A character string specifying which regression method is used, `"lm"`, `"glm"` or `"gam"`.
`family`	This is a family object specifying the distribution and link to use in fitting.
`nfolds`	Number of folds for the cross-validation procedure, for `deviance`, `R2` or `variance` criterion.
`cluster`	A logical value. If `TRUE` (default) the code is parallelized.
`ncores`	An integer value specifying the number of cores to be used in the parallelized procedure. If `NULL`, the number of cores to be used is equal to the number of cores of the machine -1.
	`test()` arguments
`x`	A data frame containing all the covariates.
`y`	A vector with the response values.
`method`	A character string specifying which regression method is used, `"lm"`, `"glm"` or `"gam"`.
`family`	This is a family object specifying the distribution and link to use in fitting.
`nboot`	Number of bootstrap repeats.
`speedup`	A logical value. If `TRUE` (default), the testing procedure is computationally efficient since it considers one more variable to fit the alternative model than the number of variables used to fit the null. If `FALSE`, the fit of the alternative model is based on considering the best subset of variables of size greater than q, the one that minimizes an information criterion. The size of this subset must be given by the user filling the argument `qmin`.
`qmin`	By default `NULL`. If `speedup` is `FALSE`, `qmin` is an integer number selected by the user. To help the selection of this argument, it is recommended to visualize the graphical output of the `plot` function and choose the number `q` which minimizes the curve.
`unique`	A logical value. By default `FALSE`. If `TRUE`, the test is performed only for one null hypothesis, given by the argument `q`.
`q`	By default `NULL`. If `unique` is `TRUE`, `q` is the integer number q of $H_0(q)$ to be tested.
`bootseed`	Seed to be used in the bootstrap procedure.
`cluster`	A logical value. If `TRUE` (default), the testing procedure is parallelized.
`ncores`	An integer value specifying the number of cores to be used in the parallelized procedure. If `NULL`, the number of cores to be used is equal to the number of cores of the machine -1.

Table 2: Arguments of `selection`, `qselection` and `test` functions.

Let t be the present time, and X_t the value obtained by the series of bi-hourly means for SO_2 at instant t (5-minute temporal instants). Setting $r = 150\ \mu g/m^3N$ as the maximum value permitted for the SO_2 concentration, and half-an-hour (6 instants) as the prediction horizon, it is of interest to predict $Y = X_{t+6}$, with the best vector of $X_l = (X_t, X_{t-1}, X_{t-2}, \ldots, X_{t-17})$. Note that one of the problems that arises is to decide which temporal instants $(X_t, X_{t-1}, X_{t-2}, \ldots, X_{t-17})$ are relevant for prediction purposes, since inclusion of all the times X_l may well degrade the overall performance of the prediction model. Based on this, we demonstrate the package capabilities using these data. An excerpt of the data frame included in the package is shown below:

```
> library(FWDselect)
> data(pollution)
> head(pollution)[1:2, ]
   In17  In16  In15  In14  In13  In12  In11  In10   In9   In8
1  3.02  3.01  3.01  3.01  3.01  3.03  3.03  3.03  3.03  3.03
2 16.49 16.55 16.42 16.35 16.56 16.75 16.74 16.72 16.63 16.53
    In7   In6   In5   In4   In3   In2   In1   In0   InY
1  3.03  3.03  3.03  3.03  3.03  3.03  3.03  3.03 10.78
2 16.32 16.08 15.77 15.47 14.81 14.30 13.70 13.35 10.65
```

The variables from `In17` to `In0` correspond to the registered values of SO_2 at a specific temporal instant. `In0` denotes the zero instant (X_t), `In1` corresponds to the 5-min temporal instant before (X_{t-1}), `In2` is the 10-min temporal instant before (X_{t-2}), and so on until the last variable. The last column of the data frame (`InY`) refers to the response variable, $Y = X_{t+6}$, the temporal instant that we wish to predict. For this purpose, we propose the underlying generalised additive model

$$Y = m_0(X_t) + m_1(X_{t-1}) + \ldots + m_{17}(X_{t-17}) + \varepsilon \tag{5}$$

where m_j, with $j = 0, \ldots, 17$, are smooth and unknown functions and ε is the error which is assumed to have mean zero. To estimate the model in (5), **FWDselect** allows penalised regression splines, implemented in the **mgcv** package (Wood, 2003, 2004, 2011).

It may often be of interest to determine the best subset of variables of size q needed to predict the response. The question that naturally arises in this application is, what is the best temporal instant for predicting an emission episode. This is easy to ascertain with the function `selection` and the argument `q = 1`. Also, based on the model that we want to estimate here (additive model), we have to use "`gam`" on the `method` argument.

```
> x <- pollution[, -19]
> y <- pollution[, 19]
> obj1 <- selection(x, y, q = 1, method = "gam", criterion = "deviance")
> obj1

****************************************************
Best subset of size q = 1 : In0

Information Criterion Value - deviance : 278663
****************************************************
```

For more than one subset size, the `qselection` function returns a table for the different subset sizes, with the selected variables and the information criterion value.

```
> obj2 <- qselection(x, y, qvector = c(1:6), method = "gam", criterion = "deviance")
[1] "Selecting subset of size 1 ..."
[1] "Selecting subset of size 2 ..."
[1] "Selecting subset of size 3 ..."
[1] "Selecting subset of size 4 ..."
[1] "Selecting subset of size 5 ..."
[1] "Selecting subset of size 6 ..."
> obj2
  q   deviance                    selection
1 1 278662.959                          In0
2 2 201474.673                     In0, In2
3 3 232164.509                In0, In2, In1
4 4 219941.426           In0, In3, In1, In5
5 5 184293.934      In0, In3, In1, In7, In6
6 6 200877.902 In0, In3, In1, In7, In6, In5
```

The above function output is a useful display that greatly helps determine the most relevant variables. A plot of this object can easily be obtained by using the following input command:

```
> plot(obj2)
```

Figure 1 shows the deviance values (obtained by cross-validation) corresponding to the different subsets. In each subset, q represents the number of temporal instants included in the model. Note, however, that only the results until subset of size $q = 6$ are shown because, from this size onwards, the rest of the obtained models have similar deviances.

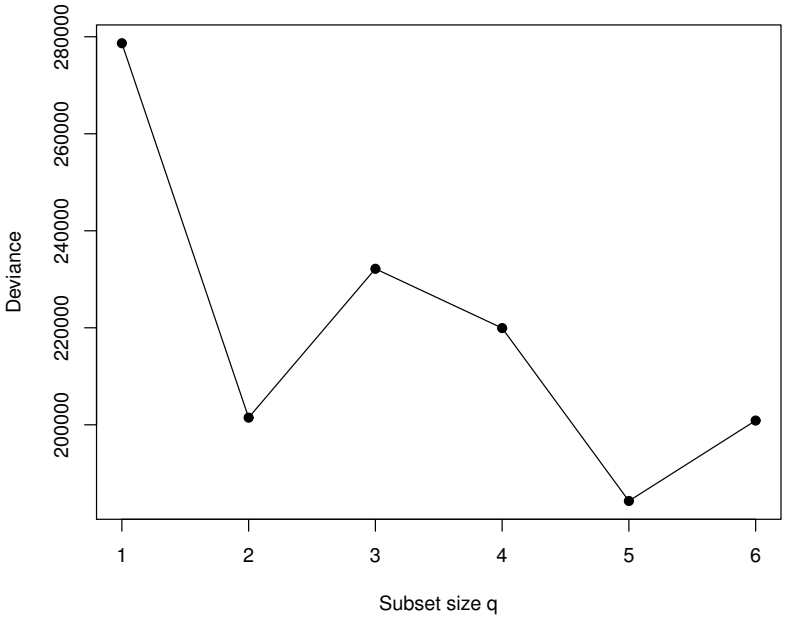


Figure 1: For each subset of size q, cross-validation deviance obtained by the best model for the pollution data.

The performance of the proposed predictors was then evaluated in a new real pollution episode. We estimate firstly each of the proposed models using the `gam` function of the `mgcv` package with the training data set (`pollution` data). Then, we apply the `predict.gam` function to each model using, in this case, the test data set. These data are found in the `episode` data, also included in this package. The corresponding data frame is illustrated as follows:

```
> data(episode)
> head(episode)[1:2, ]
  In17 In16 In15 In14 In13 In12 In11 In10  In9  In8  In7  In6  In5
1 3.02 3.02 3.03 3.10 3.10 3.10 3.10 3.22 3.27 3.33 3.36 3.38 3.47
2 3.02 3.03 3.10 3.10 3.10 3.10 3.22 3.27 3.33 3.36 3.38 3.47 3.50
   In4  In3  In2  In1  In0  InY  time
1 3.50 3.56 3.61 4.28 4.60 5.45 00:00
2 3.56 3.61 4.28 4.60 4.68 6.20 00:05
```

The course of the incident is depicted in Figure 2. Temporal instants are plotted on the horizontal axis and the real 2-hour mean SO_2 concentration that we seek to predict ($Y = X_{t+6}$) is represented by a grey line. The predictions obtained by applying the different models are shown in the same figure. The code, both for the predictions as for the plot, is shown in the Appendix.

The prediction obtained with the inclusion of just one variable in the model, X_t, is far from the optimum. However, the addition of one more variable, X_{t-2}, resulted in a remarkable increase in the model predictive capability. It makes possible for predictions close to real values to be obtained. Lastly, it can be seen that the incorporation of one more variable or temporal instant (X_{t-1}) in the model does not produce any improvement in pollution-incident prediction. Numerically speaking, the same results can be observed by taking into account the Mean Square Error for each model (Table 3).

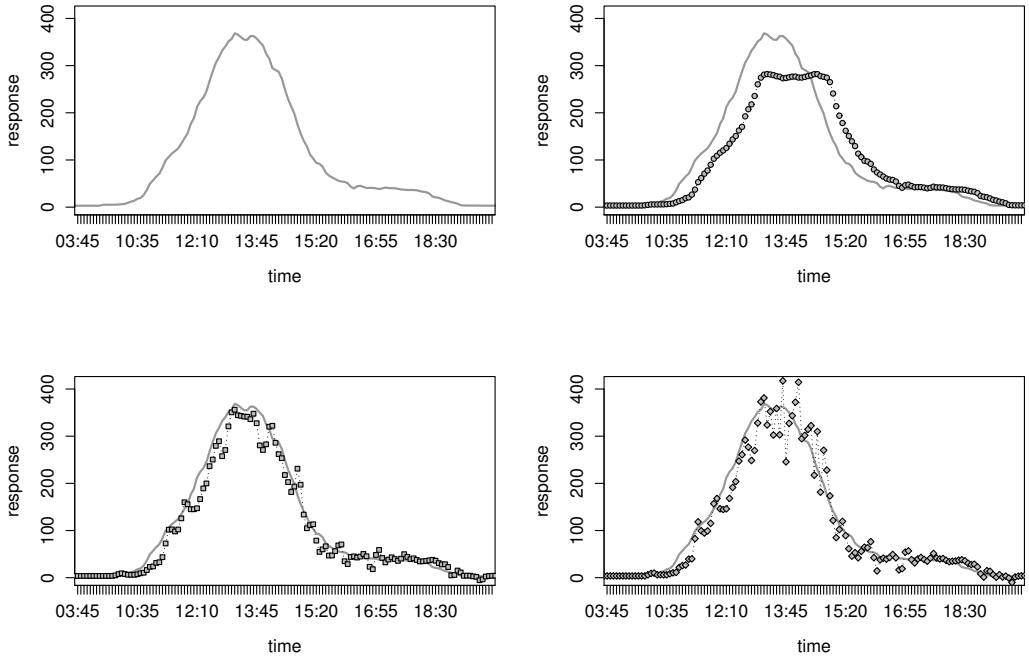


Figure 2: Example of an SO_2 pollution incident that occurred on 4 July 2003. Temporal instants are shown on the horizontal axis. The grey line represents the known response of SO_2 levels in $\mu g/m^3N$. Estimation of SO_2 levels with one, two and three covariates are represented by circles, squares and diamonds respectively.

Table 3: Mean Square Error of the selected models.

Model	MSE
$Y = X_t$	1 682.14
$Y = X_t + X_{t-2}$	366.44
$Y = X_t + X_{t-2} + X_{t-1}$	556.49

The question that now arises is what is the minimum number of variables that must be used in order to obtain the best prediction. It is possible to deduce that there is an optimal intermediate point between the number of variables that enters the model (preferably low) and the deviance value (preferably also low). To find this point, the test described in the previous section for the null hypothesis $H_0(q)$ is applied for each size, q (through the input command shown below). The procedure stops when a certain null hypothesis is accepted. The most computationally demanding parts are those that involve the bootstrap and the cross-validation techniques. This can be parallelized using the argument `cluster = TRUE` (default). This should considerably increase the performance on multi–core / multi–threading machines.

```
> test(x, y, nboot = 100, method = "gam", bootseed = 0413)
[1] "Processing IC bootstrap for H_0 ( 1 )..."
[1] "Processing IC bootstrap for H_0 ( 2 )..."

************************************
  Hypothesis Statistic pvalue     Decision
1    H_0 (1)   5779.03      0     Rejected
2    H_0 (2)    959.21   0.78 Not Rejected
```

The deduction to be drawn is that, for a 5% significance level, the null hypothesis is rejected with $q = 1$ and accepted thereafter. From these results, it can be concluded that the best temporal instants for prediction of an emission episode would be X_t and X_{t-2}.

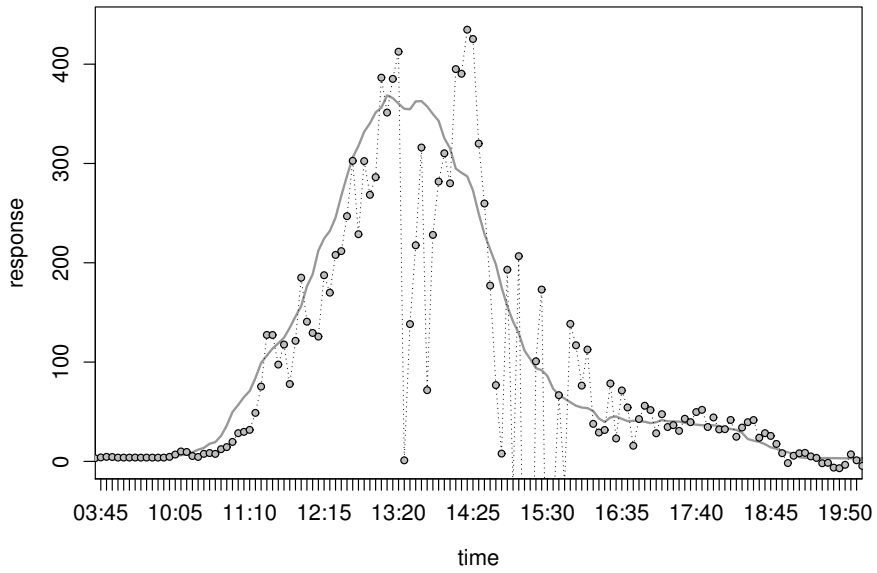


Figure 3: Example of an SO_2 pollution incident that occurred on 4 July 2003. Temporal instants are shown on the horizontal axis. The grey line represents the known response of SO_2 levels in $\mu g/m^3N$. Estimation of SO_2 levels obtained by means of the double penalty GAM is represented by circles.

Lastly, as we mention before, there are other alternatives for variable selection in additive models. One of the best-known and used procedures is the argument `select` of the `gam` function from the **mgcv** package (Marra and Wood, 2011). To illustrate and compare its usage with our procedure, we have estimated the model in (5) by means of the cited function using the `pollution` data. Then, its performance was evaluated using again the `episode` data. The prediction obtained using this double penalty GAM is far from what it should be (see Figure 3), actually, the mean square error obtained (5 024.29) is the worst of all so far (see the code in Appendix). It seems that, in situations with a large number of variables, the selection of the best subset could be a better approach.

Conclusions

This paper discusses implementation in R of a new algorithm for the problem of variable selection in a regression framework. The **FWDselect** package provides R users a simple method for ascertaining the relevant variables for prediction purposes and how many of these should be included in the model. The proposed method is a new forward stepwise-based selection procedure that selects a model containing a subset of variables according to an information criterion, and also takes into account the computational cost. Bootstrap techniques have been used to determine the minimum number of variables needed to obtain an appropriate prediction.

In some situations, several statistically equivalent optimal models of size q may exist. In such cases, **FWDselect** allows the user to visualise those models and select the most interesting one. This is obtained with the argument `seconds = TRUE` of the `selection` functions. In addition, the software provides the user with a way of easily obtaining the best subset of variables using different types of data in different frameworks, by applying the `lm`, `glm` and `gam` functions already implemented in R. The use of these classical R functions nevertheless entails a high computational cost. Hence, a further interesting extension would be the implementation of this package using C, C++ or Fortran as the programming language. R users could profit from this advantage in a future version of this package.

Insofar as the validity of the method is concerned, we think that the results obtained with simulated data are correct, and the results with the diabetes data are in accordance with other methodologies. This suggest that the behavior of the procedure in a nonparametric framework will be also adequate.

The results in this paper were obtained using *R* 3.2.0. The **FWDselect** package (Sestelo et al., 2015) is available from the Comprehensive R Archive Network at the URL `http://cran.r-project.org/web/packages/FWDselect/`.

Acknowledgments

This work was supported by research grant SFRH/BPD/93928/2013 of "Fundação para a Ciência e a Tecnologia" (FCT) and by FEDER Funds through "Programa Operacional Factores de Competitividade - COMPETE", by Portuguese Funds through FCT, in the form of grant PEst-OE/MAT/UI0013/2014, by grant MTM2011-23204 (FEDER support included) of the Spanish Ministry of Science and Innovation and by grant 10PXIB300068PR from the Galician Regional Authority (Xunta de Galicia).

Bibliography

H. Akaike. Information theory and an extension of the maximum likelihood principle. In *Second International Symposium on Information Theory*, pages 267–281. Akademiai Kiado, 1973. [p132]

C. Belitz and S. Lang. Simultaneous selection of variables and smoothing parameters in structured additive regression models. *Computational Statistics & Data Analysis*, 53(1):61 – 81, 2008. doi: http://dx.doi.org/10.1016/j.csda.2008.05.032. URL http://www.sciencedirect.com/science/article/pii/S0167947308003009. [p133]

A. Buja, T. Hastie, and R. Tibshirani. Linear smoothers and additive models. *The Annals of Statistics*, 17: 453–510, 1989. [p133]

V. Calcagno. glmulti: Model selection and multimodel inference made easy. R package version 1.0.7, 2013. URL http://cran.r-project.org/web/packages/glmulti/index.html. [p133]

V. Calcagno and C. de Mazancourt. glmulti: An R package for easy automated model selection with (Generalized) Linear Models. *Journal of Statistical Software*, 34(12):1–29, 2010. [p132]

B. Efron. Bootstrap methods: another look at the jackknife. *The Annals of Statistics*, 7:1–26, 1979. [p136]

E. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall, London, 1993. [p136]

J. Fan and J. Jiang. Nonparametric inferences for additive models. *Journal of the American Statistical Association*, 100:890–907, 2005. URL http://ideas.repec.org/a/bes/jnlasa/v100y2005p890-907.html. [p133]

J. Fan and J. Jiang. Nonparametric inference with generalized likelihood ratio tests. *TEST*, 16(3):409–444, 2007. doi: 10.1007/s11749-007-0080-8. URL http://dx.doi.org/10.1007/s11749-007-0080-8. [p133]

J. Fan, C. Zhang, and J. Zhang. Generalized likelihood ratio statistics and wilks phenomenon. *The Annals of Statistics*, 29(1):153–193, 2001. ISSN 00905364. doi: 10.2307/2674021. URL http://dx.doi.org/10.2307/2674021. [p133]

M. R. Forster. Key concepts in model selection: Performance and generalizability. *Journal of Mathematical Psychology*, 44(1):205–231, 2000. [p132]

J. Friedman, T. Hastie, and R. Tibshirani. *glmnet: Lasso and Elastic-Net Regularized Generalized Linear Models*, 2015. URL http://CRAN.R-project.org/package=glmnet. R package version 2.0-2. [p133]

G. M. Furnival and R. W. Wilson. Regressions by leaps and bounds. *Technometrics*, 16(4):499–511, 1974. doi: 10.1080/00401706.1974.10489231. URL http://www.tandfonline.com/doi/abs/10.1080/00401706.1974.10489231. [p132]

I. García-Jurado, W. González-Manteiga, J. M. Prada-Sánchez, M. Febrero-Bande, and R. Cao. Predicting using Box-Jenkins, nonparametric, and bootstrap techniques. *Technometrics*, (3):303–310. [p137]

P. J. Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82:711–732, 1995. [p132]

C. Hans. Bayesian lasso regression. *Biometrika*, 96(4):835–845, 2009. doi: 10.1093/biomet/asp047. URL http://biomet.oxfordjournals.org/content/96/4/835.abstract. [p132]

W. Härdle and P. Hall. On the backfitting algorithm for additive regression models. *Statistica Neerlandica*, 47(1):43–57, 1993. doi: 10.1111/j.1467-9574.1993.tb01405.x. URL http://dx.doi.org/10.1111/j.1467-9574.1993.tb01405.x. [p133]

W. Härdle and E. Mammen. Comparing nonparametric versus parametric regression fits. *The Annals of Statistics*, 21(4):1926–1947, 1993. [p136]

T. Hastie and B. Efron. *lars: Least Angle Regression, Lasso and Forward Stagewise*, 2013. URL http://CRAN.R-project.org/package=lars. R package version 1.2. [p132]

T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, 2003. [p132]

T. J. Hastie and D. Pregibon. Generalized linear models. In J. M. Chambers and T. J. Hastie, editors, *Statistical Models in S*, page 335. Wadsworth & Brooks/Cole, 1992. [p133]

R. R. Hocking. A biometrics invited paper. the analysis and selection of variables in linear regression. *Biometrics*, 32(1):1–49, 1976. doi: 10.2307/2529336. URL http://dx.doi.org/10.2307/2529336. [p132]

G. Kauermann and J. Opsomer. Local likelihood estimation in generalized additive models. *Scandinavian Journal of Statistics*, 30:317–337, 2003. [p136]

L. Kuo and B. Mallick. Variable selection for regression models. *The Indian Journal of Statistics (Special Issue on Bayesian Analysis)*, 60(B):65–81, 1998. [p132]

Y. Lin and H. H. Zhang. Component selection and smoothing in multivariate nonparametric regression. *The Annals of Statistics*, 34(5):2272–2297, 2006. doi: 10.1214/009053606000000722. URL http://dx.doi.org/10.1214/009053606000000722. [p133]

R. Y. Liu. Bootstrap procedures under some non-i.i.d. models. *The Annals of Statistics*, 16(4):1696–1708, 1988. URL http://www.jstor.org/stable/2241788. [p136]

T. Lumley and A. Miller. leaps: regression subsets selection. R package version 2.9, 2009. URL http://CRAN.R-project.org/package=leaps. [p132]

C. L. Mallows. Some comments on cp. *Technometrics*, 15(4), 1973. doi: 10.2307/1267380. URL http://dx.doi.org/10.2307/1267380. [p132]

E. Mammen. Bootstrap and wild bootstrap for high dimensional linear models. *The Annals of Statistics*, 21(1):255–285, 1993. URL http://www.jstor.org/stable/3035590. [p136]

E. Mammen, O. Linton, and J. Nielsen. The existence and asymptotic properties of a backfitting projection algorithm under weak conditions. *The Annals of Statistics*, 27:1443–1490, 1999. [p133]

G. Marra and S. N. Wood. Practical variable selection for Generalized Additive Models. *Computational Statistics & Data Analysis*, 55(7):2372–2387, 2011. doi: 10.1016/j.csda.2011.02.004. URL http://dx.doi.org/10.1016/j.csda.2011.02.004. [p133, 142]

A. I. Mcleod and C. Xu. bestglm: Best subset GLM. R package version 0.34, 2014. URL http://CRAN.R-project.org/package=bestglm. [p133]

A. Miller. *Subset Selection in Regression*. Chapman and Hall/CRC, Boca Raton, FL, 2002. [p132]

J. Neyman and E. S. Pearson. On the use and interpretation of certain test criteria for purposes of statistical inference: part II. *Biometrika*, 20A(3/4):263–294, 1928. doi: 10.2307/2332112. URL http://dx.doi.org/10.2307/2332112. [p133]

J. Orestes Cerdeira, A. P. Duarte Silva, J. Cadima, and M. Minhoto. subselect: Selecting variable subsets. R package version 0.12-5, 2015. URL http://CRAN.R-project.org/package=subselect. [p132]

T. Park and G. Casella. The Bayesian Lasso. *Journal of the American Statistical Association*, 103(482): 681–686, 2008. [p132]

J. M. Prada-Sánchez and M. Febrero-Bande. Parametric, non-parametric and mixed approaches to prediction of sparsely distributed pollution incidents: a case study. *Journal of Chemometrics*, 11(1): 13–32, 1997. [p137]

J. M. Prada-Sánchez, M. Febrero-Bande, T. Cotos-Yáñez, W. González-Manteiga, J. L. Bermúdez-Cela, and T. Lucas-Domínguez. Prediction of SO2 pollution incidents near a power station using partially linear models and an historical matrix of predictor-response vectors. *Environmetrics*, 11(2):209–225, 2000. [p137]

J. Roca-Pardiñas, W. González-Manteiga, M. Febrero-Bande, J. M. Prada-Sánchez, and C. Cadarso-Suárez. Predicting binary time series of SO2 using Generalized Additive Models with unknown link function. *Environmetrics*, 15(7):729–742, 2004. [p137]

J. Roca-Pardiñas, C. Cadarso-Suárez, P. G. Tahoces, and M. J. Lado. Selecting variables in non-parametric regression models for binary response. An application to the computerized detection of breast cancer. *Statistics in Medicine*, 28(2):240–259, 2009. [p134]

G. A. F. Seber. *Linear Regression Analysis*. Wiley, 1997. [p133]

G. A. F. Seber and C. Wild. *Nonlinear Regression*. Wiley, 1989. [p133]

M. Sestelo. *Development and computational implementation of estimation and inference methods in flexible regression models. Applications in Biology, Engineering and Environment*. PhD thesis, Department of Statistics and O.R., University of Vigo, 2013. [p136]

M. Sestelo, N. M. Villanueva, and J. Roca-Pardiñas. Fwdselect: Selecting variables in regression models. R package version 2.1.0, 2015. URL http://cran.r-project.org/web/packages/FWDselect/index.html. [p142]

R. Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society Series B*, 58(1):267–288, 1996. [p132]

G. Tutz and H. Binder. Generalized additive modeling with implicit variable selection by likelihood-based boosting. *Biometrics*, 62(4):961–971, 2006. doi: 10.1111/j.1541-0420.2006.00578.x. URL http://dx.doi.org/10.1111/j.1541-0420.2006.00578.x. [p133]

W. N. Venables and B. D. Ripley. *Modern applied statistics with S-Plus*. Springer, second edition, 1997. [p132]

H. Wickham. meifly: Interactive model exploration using GGobi. R package version 0.3, 2014. URL http://CRAN.R-project.org/package=meifly. [p132]

S. N. Wood. Thin plate regression splines. *Journal of the Royal Statistical Society - Series B: Statistical Methodology*, 65(1):95–114, 2003. [p139]

S. N. Wood. Stable and efficient multiple smoothing parameter estimation for Generalized Additive Models. *Journal of the American Statistical Association*, 99(467):673–686, 2004. [p139]

S. N. Wood. *Generalized Additive Models: An Introduction with R*. Chapman & Hall/CRC, Boca Raton, FL, 2006. URL http://cran.r-project.org/src/contrib/Descriptions/gamair.html. ISBN 1-584-88474-6. [p133]

S. N. Wood. Fast stable restricted maximum likelihood and marginal likelihood estimation of semi-parametric Generalized Linear Models. *Journal of the Royal Statistical Society Series B*, 73(1):3–36, 2011. [p133, 139]

C. F. J. Wu. Jackknife, bootstrap and other resampling methods in regression analysis. *The Annals of Statistics*, 14(4):1261–1295, 1986. doi: 10.2307/2241454. URL http://dx.doi.org/10.2307/2241454. [p136]

Marta Sestelo
Centre of Mathematics, University of Minho, Portugal
SiDOR Research Group and CINBIO, University of Vigo, Spain
sestelo@uvigo.es

Nora M. Villanueva
Department of Statistics and O.R., University of Vigo, Spain
nmvillanueva@uvigo.es

Luis Meira-MachadoAuthor
Centre of Mathematics, University of Minho, Portugal
lmachado@math.uminho.pt

Javier Roca-Pardiñas
Department of Statistics and O.R., University of Vigo, Spain
roca@uvigo.es

Appendix

Here we illustrate the use of **FWDselect** using simulated data. The use of simulated data allows us to ascertain the behavior of our software and to compare it with others available tools designed for the same purpose.

Consider a vector of 10 covariates, $\mathbf{X} = (X_1, \ldots, X_{10})$, and a continuous response, Y, generated in accordance with

$$Y = 2\ X_1 + 4\ X_5 + \varepsilon,$$

where ε is the error distributed in accordance with a $N(0,1)$ and the explanatory covariates are random variables obtained from uniforme distribution on $[-1,1]$. Note that we have a linear scenario in which the response variable depends only on two covariates.

The following code will simulate 100 observations ($n = 100$) from the above model. To ensure the reproducibility of the results reported in the paper a seed was considered (0413).

```
> library(glmulti)
Loading required package: rJava
> library(leaps)
> rm(list = ls())
> set.seed(0413)
> n <- 100
> x <- matrix(runif(10 * n, -1, 1), ncol = 10, nrow = n, byrow = FALSE)
> e <- rnorm(n, 0, 1)
> y <- 2 * x[, 1] + 4 * x[, 5] + e
> data <- data.frame(x, y)
```

Now we compare our method against other existing methodologies developed to perform automated variable selection. We choose the **leaps** package (`regsubsets` function), which selects the best variables for each subset of size q without determining the number of variables that users have to include in the model; the `step` function from the **stats** package which selects a formula-based model using the AIC; and the **glmulti** which compares all posible models through an exhaustive screening of the candidates, or a genetic algorithm, or a very fast exhaustive branch-and-bound algorithm .

```
> res1 <- regsubsets(x, y)
> summary(res1)$outmat[2, ]
  a   b   c   d   e   f   g   h   i   j
"*" " " " " " " " "*" " " " " " " " " " "

> res2 <- step(lm(y ~ ., data = data), trace = FALSE)
> res2

Call:
lm(formula = y ~ X1 + X4 + X5 + X9, data = data)

Coefficients:
(Intercept)           X1           X4           X5           X9
     0.1538       1.9691       0.4285       3.5774       0.2875


> res3 <- glmulti(y ~ ., data = data, level = 1, plotty = F, report = FALSE)
> summary(res3)$bestmodel
[1] "y ~ 1 + X1 + X4 + X5 + X9"

> res4aux <- test(x, y, nboot = 100)
[1] "Processing IC bootstrap for H_0 ( 1 )..."
[1] "Processing IC bootstrap for H_0 ( 2 )..."

************************************
  Hypothesis Statistic pvalue     Decision
1    H_0 (1)     90.92      0     Rejected
2    H_0 (2)     21.11   0.06 Not Rejected
> res4 <- selection(x = x, y = y, q = res4aux$nvar, cluster = FALSE)
> res4
```

```
**************************************************
Best subset of size q = 2 : 5 1

Information Criterion Value - deviance : 16.70488
**************************************************
```

The regsubsets function is based on all subsets or, in other words, exhaustive variable selection. The method identifies the best subsets of linear predictors using a branch–and–bound algorithm (Miller, 2002). Since this function returns separate best models of all sizes, we consider only the results obtained for a subset of size two. In this case, the procedure works properly returning the X_1 and X_5 variables as the best subset of size two. The model–selection oriented function step is a widely used methodology for jointly determining the number and choice of variables. In this case, this procedure fails returning a model which includes the effects of four covariates (X_1, X_4, X_5 and X_9). The results obtained with the **glmulti** package, another option for model selection, are also mistaken. The procedure returns the same model obtained with the previous method (step). Finally, in order to ascertain the performance of **FWDselect**, we firstly apply the test function with the purpose of determine the number of variable that have to be included in the model. Then, once this number is obtained (saved in the returned list as $nvar), the selection function determines correctly the X_1 and X_5 variables.

According to the computation time of these four methods, the fastest procedure is the implemented in the **leaps** package taking only 0.001 secs. The second one is the step function which runs in 0.037 secs. The next one is the glmulti function which takes 3.149 secs. Lastly, the most computationally demanding code is the implemented in the **FWDselect** package which requires 9.181 secs. All the results have been obtained using the R's system.time command on a 2.4 GHz Intel Core i5, with 4 cores and 4 Gb of RAM.

The previous results have been obtained using simulated data with a linear effect of the covariates. However, in practice, the user does not know the dependence structure, i. e., how the response variable depends on the covariates. With this in mind, we have considered and applied again the four procedures on another scenario where the response variable depends again on the same two covariates, but in this case, the effect of them is nonlinear. Particularly, the Y is now generated in accordance with

$$Y = 2\,(X_1)^2 + 2\sin(2\pi X_5) + \varepsilon,$$

being both ε and the explanatory covariates the same of the previous scenario. Note that we have now a nonlinear scenario in which the response variable depends only on two covariates.

```
> y <- 2 * x[, 1]**2 + 2 * sin(2 * pi * x[, 5]) + e
> data <- data.frame(x, y)
> res1 <- regsubsets(x, y)
> summary(res1)$outmat[2, ]
  a   b   c   d   e   f   g   h   i   j
" " " " " " " " "*" "*" " " " " " " " "
```

```
> res2 <- step(lm(y ~ ., data = data), trace = 0)
> res2

Call:
lm(formula = y ~ X5, data = data)

Coefficients:
(Intercept)           X5
     0.4764      -1.2377
```

```
> res3 <- glmulti(y ~ ., data = data, level = 1, plotty = F, report = FALSE)
> summary(res3)$bestmodel
[1] "y ~ 1 + X5"
```

```
> res4aux <- test(x, y, nboot = 100, method = "gam")
[1] "Processing IC bootstrap for H_0 ( 1 )..."
[1] "Processing IC bootstrap for H_0 ( 2 )..."
```

```
************************************
  Hypothesis Statistic pvalue     Decision
1    H_0 (1)     46.04      0     Rejected
2    H_0 (2)     20.89   0.15 Not Rejected
> res4 <- selection(x, y, q = res4aux$nvar, cluster = FALSE, method = "gam")
> res4

************************************************
Best subset of size q = 2 : 5 1

Information Criterion Value - deviance : 18.41203
************************************************
```

In this case, the performance of the methods changes. Excluding the **FWDselect**, all the procedures fail to select the correct model. The **leaps** package returns the X_5 and X_6 variables whereas the others two packages only retrieve the effect of X_5.

The results presented in this appendix have been obtained with one simulated sample of $n = 100$. In order to evaluate the real performance of the methods, a simulation study using five hundred independent samples with different sample sizes ($n = 50, 100, 200$) was carried out. Focusing on the linear scenario, the **leaps** and **FWDselect** packages work well with 100% and close to 95% of successes, respectively (for any sample sizes). The success rate for the other two packages is around 22%. Note that the results of **leaps** have been obtained assuming a subset of size two, and thus providing an advantage to this method over the others. In relation with the nonlinear scenario, the proportion of failures is very high for all procedures excepting **FWDselect**. The latter performs correctly close to 30% of the times for the smallest sample size, around 63% for $n = 100$ while it reaches 91.6% of successes for $n = 200$.

An Interactive Survey Application for Validating Social Network Analysis Techniques

by Mitchell Joblin and Wolfgang Mauerer

Abstract Social network analysis is extremely well supported by the R community and is routinely used for studying the relationships between people engaged in collaborative activities. While there has been rapid development of new approaches and metrics in this field, the challenging question of validity (how well insights derived from social networks agree with reality) is often difficult to address. We propose the use of several R packages to generate interactive surveys that are specifically well suited for validating social network analyses. Using our web-based survey application, we were able to validate the results of applying community-detection algorithms to infer the organizational structure of software developers contributing to open-source projects.

Introduction

Social network analysis (SNA) is an increasingly popular approach to study the relationships between individuals engaged in collaborative activities (Ahn et al., 2007; Mislove et al., 2007; Kumar et al., 2010), and numerous high quality R packages support the thriving SNA community (e.g., **igraph**, **sna**, **graph**, **twitteR**, **Rfacebook**, etc.; Csardi and Nepusz 2006; Butts 2014; Gentleman et al.; Gentry 2015; Barbera and Piccirilli 2015). What is often not clear is the validity of SNA approaches that propose new metrics or apply existing metrics to a new source of data. In the literature, researchers have questioned and criticized studies using SNA because it is unclear if the results are reflective of reality (Donath and Boyd, 2004; Wilson et al., 2009). We developed a web-based survey application for conducting interactive surveys that specifically addresses the unique needs of the SNA community and successfully deployed the application to study the collaborative relationship between software developers in open-source projects and to validate the usage of unsupervised machine learning algorithms to infer the developers' organizational structure (Joblin et al., 2015).

In social network analysis, the relationships between individuals are formalized as a graph where nodes represent people and the edges between nodes represent a particularly interesting connection. For example, Twitter data can be used to construct a retweet network where an edge between individuals exists if one individual has retweeted another individual's tweet. The particular heuristic used to establish an edge between individuals is chosen based on the desired concept to study. For example, a retweet may indicate endorsement of the message being tweeted. From this, one could conclude that users with many retweets of their content are regarded as an influential person within that local group of people. One of the primary challenges with this style of analysis is validating whether the assumptions about the relationship heuristic are correct. It may, for instance, not be clear whether a retweet always indicates a positive sentiment. Alternatively, retweets could also stem from controversial topics and may not be ubiquitously regarded as supportive of the original tweet's message.

While SNA is not primarily concerned with constructing social networks, but rather to analyze the network's properties, the network construction heuristic influences the validity of the subsequent analysis. In general, the goal of SNA is to identify interesting features of a social network that capture an abstract quality of social relationships. For example, finding important or highly influential actors in a network is one of the most well-researched areas of SNA where one considers the local or global network topology to identify individuals that are exceptionally well-connected to other actors. The notion of centrality, and many other network metrics, is a duality where one definition is a mathematical formalization based on the network topology and the other definition is an abstract social concept such as influence. The challenge we wish to address with our survey application is to validate the claim that the mathematical formalizations provided by the field of SNA are congruent with the abstract social concept we wish to identify.

Our survey application is designed to address the following three concerns that are fundamental to the validity of SNA:

- **Network Construction Heuristic**
  Example: Do edges in the network accurately represent relationships between actors in reality?
- **Network Structural Property**
  Example: Does the community structure of a network accurately identify subgraphs of individ-

uals with common goals or interests in reality?

- **Network Metric**
  Example: Do centrality metrics accurately represent the level of influence of an actor in reality?

All source code that implements our survey application is available at a supplementary site: `http://github.com/mitchell-joblin/SNA_survey_framework`.

Challenges

Developing and deploying a survey for SNA purposes involves a set of unique challenges and requirements that are not currently satisfied by existing survey templates and tools. We now introduce the set of requirements we identified and specifically addressed with our survey application.

Requirement 1: Ease of large scale deployment and collection of responses

Modern social networks can range in size from hundreds to millions of nodes and the survey delivery mechanism should be designed to handle deployment under large scale conditions. A web solution enables the survey participant to easily login to the web interface and submit their responses without the need to download or install any software to complete the survey. Any challenges experienced while participating in a survey create a barrier to completion and likely contribute to lower return-rates and quality responses. In additional to scalability and ease of use, a web solution also allows for aggregation of responses into a common database for later analysis.

Requirement 2: Interactivity

Performing a survey for SNA purposes will often involve the need to display a labeled graph to the survey participants. Research in graph layout and visualization is continually advancing; however, the optimal visualization and layout parameters are dependent on network properties in a non-trivial way. The network size, edge density, graph type (e.g., directed, undirected, weighted, unweighted, one-mode, and two-mode) all influence the optimal visualization. Readability of the graph is necessary for quality responses. To ensure graph details were not obscured by problems such as overlapping nodes or edges, the survey participant should be able to influence a set of visual parameters so that all necessary details of the graph are observable. The adjustable parameters also allow the visually impaired to participate more effectively.

Requirement 3: Dynamic survey content generation

We determined that certain elements of the survey needed to be generated dynamically so that each survey participant would be shown information that was relevant to their particular position in the network. We identified each survey participant through a login process and then computed relevant data such as the subgraph community they were found in and the set of people which we expected to be influential to them. We found this to be a particularly powerful and interesting aspect of our survey because the responses often provided insights about the network that would not have been obvious if we had not shown the relevant network data and instead only asked general questions.

Requirement 4: Integration with existing R infrastructure

One of our primary concerns with developing the survey was the expenditure of effort to prepare our existing SNA analysis pipeline for use in the survey instrument. A substantial amount of support for SNA already exists in R (e.g., **igraph**, **sna**, **graph** etc.), therefore it is highly desirable to seamlessly and effortlessly integrate existing R infrastructure into the survey application. By taking advantage of the Shiny R web application framework, we could avoid a substantial amount of effort to adapt existing R infrastructure to another language or platform for the survey deployment.

Requirement 5: Visually appealing and professional aesthetic

In a preliminary analysis of options for survey platforms, we realized that many of the existing tools did not support a visually appealing or professional aesthetic. We felt that an unprofessional appearance would compromise the seriousness and credibility of the organization hosting the survey and deter survey participation. Perhaps potential survey candidates would perceive the survey poorly

and think that the organization would mishandle the collected data for unethical reasons or via poor execution such that the results would be useless and answering the questions would be futile.

Alternative survey tools

A number of survey tools are available online such as SurveyMonkey (www.surveymonkey.com) and LimeSurvey (www.limesurvey.org), but we found that these tools were not capable of satisfying the requirements for validating SNA techniques. In a canonical survey, a number of predetermined questions are presented to the survey participants and the responses are predetermined categories or free text fields. In the case of predominantly static and predetermined survey content, the features offered by the above tools are more than adequate and customizable. The inadequacies of these tools stem from the lack of features for interfacing with R infrastructure and supporting interactive survey content. Both SurveyMonkey and LimeSurvey have convenient import features that provide a mechanism to display precomputed survey content. Prior to developing our own application, we considered precomputing the survey content for all the possible survey participants and then using the import feature. The problem with this approach was that we did not have a-priori knowledge of the required content and computing all possible variations of the survey content would be incredibly wasteful. When conducting a survey, one typically expects roughly a 10% response rate so computing the necessary data for all potential participants would be roughly 90% waste. This consideration is especially important for researchers working with big data, where there may be potentially millions of survey participants. Furthermore, using this approach would not allow the survey participant to configure any visualization parameters. We found the reactive programming model provided by **shiny** (Chang et al., 2015) to be far more powerful for creating interactive surveys compared to those provided by the alternative survey tools. The added benefit of using **shiny** is that any visualization generated by an R script can be easily converted into a dynamic survey element with just a few lines of code. In contrast, managing a set of precomputed visualization requires potentially vast quantities of storage space and a schema for uniquely identifying the images to be displayed correctly in the survey.

Shiny web application framework

Shiny is a web application framework for R that allows one to easily transform their existing R code into an interactive web application. By using Shiny, we were able to quickly implement a web-based survey instrument without the need to significantly alter our existing R infrastructure for social network analysis. To build a Shiny web application two main components need to be implemented, a 'server.R' file which constructs the R objects to be displayed by the application and a 'ui.R' file to control layout and appearance. Alternatively, one can choose to implement an interface using HTML and CSS to achieve greater flexibility and customization. An example survey question taken from our application is provided in Figure 1. In the following subsections, we introduce the basic elements for implementing the example question including the creation of interactive visualizations using the **shiny** and **igraph** packages.

Example server R script

In our particular implementation we stored the user data in a relational database, as will likely be common in many applications. Below we illustrate the implementation to retrieve a specific person's ego network in the form of an edge list data frame from a MySQL database. The **igraph** package for network analysis is then used to construct the graph object from the edge list data frame and then finally plot it. Using this basic template, one can insert their own network analysis algorithms for a specific purpose. The reactive mechanism for retrieving the interactive user input from the UI is also illustrated for the vertex size and vertex label visualization parameters. In the next section, we will see how the UI is implemented for these particular visualization parameters.

Shiny uses a very powerful reactive programming paradigm to couple the client and server elements to support interactivity. Using this model, *reactive values* represent values that can change over time, and *reactive expressions* represent operations that depend on the use of reactive values. The reactive expressions track the state of reactive values so whenever an update occurs, the dependent reactive expressions are re-executed. The reactive programming concept also supports the important separation of computationally intensive processes from the interactive elements to prevent lag in the user interface. We provide an example server script in Figure 2 to illustrate a basic example for generating interactive survey content using the reactive programming model.

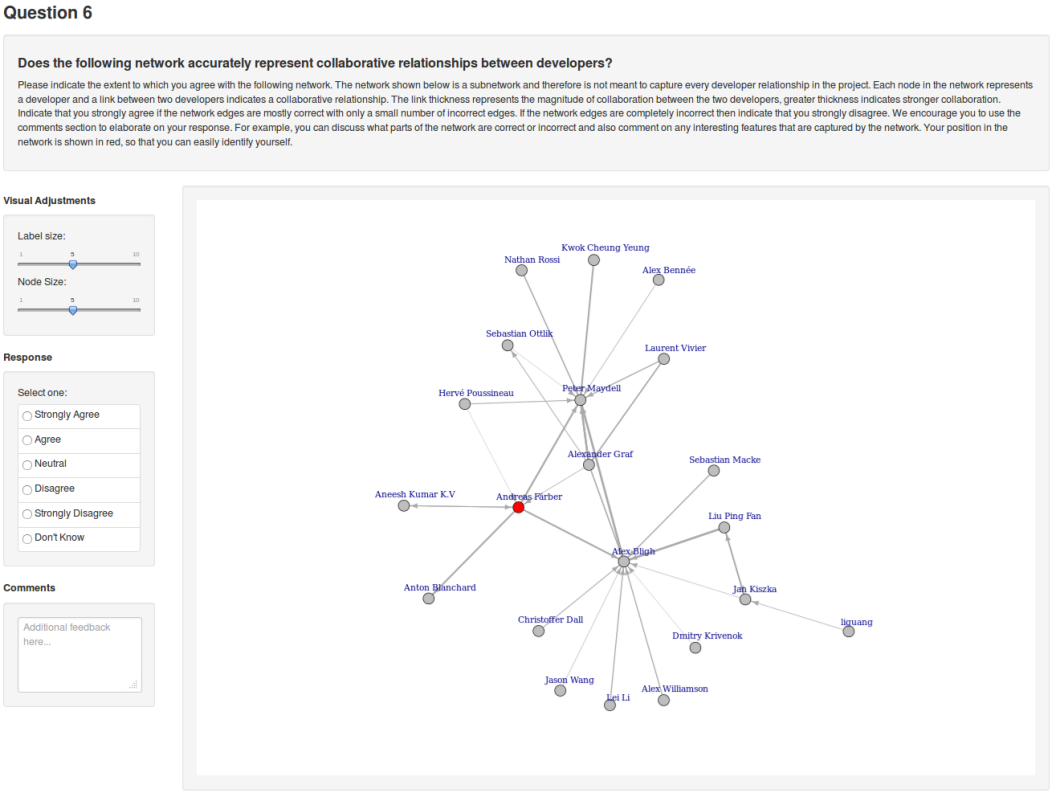


Figure 1: Example survey question.

After first executing a connection to the MySQL database, a reactive expression (`graph.data`) is defined to encapsulate the computationally intensive graph processing algorithms. The reactive expression first retrieves an ID for a specific edge list stored in the database. In example question shown in Figure 1, the ID corresponds to a specific user's community network and in the login phase description we discuss how to identify survey participants using a login process. Next, the edge list corresponding to the ID is retrieved from the database and then any computationally intensive processing is performed. Alternatively, if a database is unavailable, then an archive file or any type of storage format can be loaded into R within this reactive expression.

In the next expression, a reactive endpoint is created. Inside the expression, the reactive values that represent the vertex size (`input$vertex.size`) and label size (`input$label.size`) will cause the reactive endpoint `output$graph` to be evaluated every time an update occurs to one of the visualization parameters. A critical aspect of the implementation is the separation of the computationally intensive SNA algorithms from the visualization parameters. Without the use of reactive conductors (e.g., `graph.data`), the computationally intensive code would be re-evaluated for every update to the visualization parameters and would result in severe lag in the user interface. In the next section, we demonstrate how the binding between the elements in the server script and the user interface is established. More information about the reactive programming model used by shiny server is available in the Shiny Package Documentation.

Example HTML UI

The user interface component of the shiny web application controls the appearance and layout of the survey including all survey questions and response fields. We chose to implement the UI in HTML and CSS using the open-source framework Bootstrap, alternatively one could also implement the UI in R using **shiny**. The advantage to developing the UI in HTML is greater customizability of the look and feel, but the features provided by **shiny** are quite sufficient for most purposes and doesn't require HTML knowledge. With Bootstrap, we were able to achieve a professional aesthetic and maximum flexibility using only the basic features offered by the framework. An example survey question is shown in Figure 3 demonstrating the UI implementation in HTML to display the survey question text, an **igraph** network object with user configurable visualization parameters, and a multi-category response section. This example illustrates all the basic elements of a survey question and all the questions in our example application follow a similar format. Beginning at the top of Figure 3, the

```
library(shiny)
library(igraph)
library(RMySQL)

shinyServer(function(input, output, session, clientData) {
  # Create MySQL connection object
  con <- dbConnect(MySQL(),
                   user = 'USERNAME',
                   password = 'PASSWORD',
                   host = 'HOST',
                   dbname = 'DBNAME')

  # Query database for an edge list and perform SNA
  # and return a reactive conductor
  graph.data <- reactive({
                  # Get unique graph ID from database
                  graph.id <- get.graph.id(con)

                  # Query MySQL database for a specific network
                  edge.list <- query.graph.edges(con, graph.id)

                  # Insert any computationally intensive code for processing
                  # the graph here to avoid being recomputed
                  # for every visualization update})

  # Generate reactive endpoint
  output$graph <- renderplot({
                    edge.df <- graph.data()

                    # Get input from UI for graph label and node size
                    # from reactive sources
                    vertex.size <- input$vertex.size
                    label.size  <- input$label.size

                    # Create igraph graph object and plot
                    g <- graph.data.frame(edge.df)
                    plot(g, vertex.size = vertex.size,
                         vertex.label.cex = label.size)})
}
```

Figure 2: Server R script for example survey question.

survey question is specified inside a Bootstrap "well" element. Next, the visualization parameters for the network are specified as sliders as is shown in Figure 1. Moving further downward, the binding between the graph object provided by the 'server.R' script and the UI is made. Lastly, the response fields "agree" and "disagree" are specified as HTML form inputs. This basic example question can easily be extended to suit the needs of a wide variety of survey questions. For example, one can easily rewrite the question, change the visualization parameters, or introduce different response categories (e.g., five level Likert item). From a technical standpoint, Figure 3 clearly demonstrates how the input elements for dynamically altering the graph visualization are implemented and how the graph plot generated by the 'server.R' script is integrated into the UI.

The binding between the elements of the 'server.R' script and the UI are achieved through the variable identifiers. One can see that the HTML id tags match the corresponding variable identifiers in the 'server.R' script. In this case, the `vertex.size` visualization parameter is displayed as a slider. For categorical or other discrete parameters, drop-down menus can be used instead of a slider when needed. The appearance of the example question rendered in a browser is shown in Figure 1 and includes the graph, the basic visualization adjustments, the categorical response input, and an additional text input for comments.

The above serves as a starting point to construct more elaborate applications and easily integrate

```
<h3>Question 1</h3>
<--Survey question text-->
<div class="well">
  <h4>Does the following network accurately represent collaborative relationships?</h4>
</div>

<--Display network visualization-->
<div class="row-fluid">
  <div class="span2">
    <!--Insert all user configurable graph visualization parameters here-->
    <h5>Visual Adjustments</h5>
    <div class="well">
      <!-- Slider to change vertex size parameter, input id
               matches variable name in server.R-->
      <label class="control-label" for="vertex.size">Vertex Size:</label>
      <input class="jslider"
      data-format="#,##0.#####" data-from="1" data-locale="us" data-round="false"
      data-skin="plastic" data-smooth="false" data-step="1" data-to="10" id="vertex.size"
      name="vertex.size" type="slider" value="5">
    </div>

    <div class="span10">
      <div class="well">
        <!--Insert graph plot, id matches output variable identifier in server.R-->
        <div class="shiny-plot-output" id="graph" style=
        "width: 100%; height: 800px; margin-left:0px; margin-right:0px;
        margin-bottom:0px; margin-top:0px">
        </div>
      </div>
    </div>

    <!--Specify response fields-->
    <div>
      <h5>Response</h5>
      <div class="well">
        <div>
          <label>Select one:</label>
          <table class="table table-condensed table-bordered" style=
          "background-color:white;margin-bottom:0px">
            <tbody>
              <tr>
                <td><label class="radio"><input name="q1a" type="radio" value="agree">
                Agree</label></td>
              </tr>
              <tr>
                <td><label class="radio"><input name="q1a" type="radio" value="disagree">
                Disagree</label></td>
              </tr>
            </tbody>
          </table>
        </div>
      </div>
    </div>
  </div>
</div>
```

Figure 3: HTML UI for example survey question.

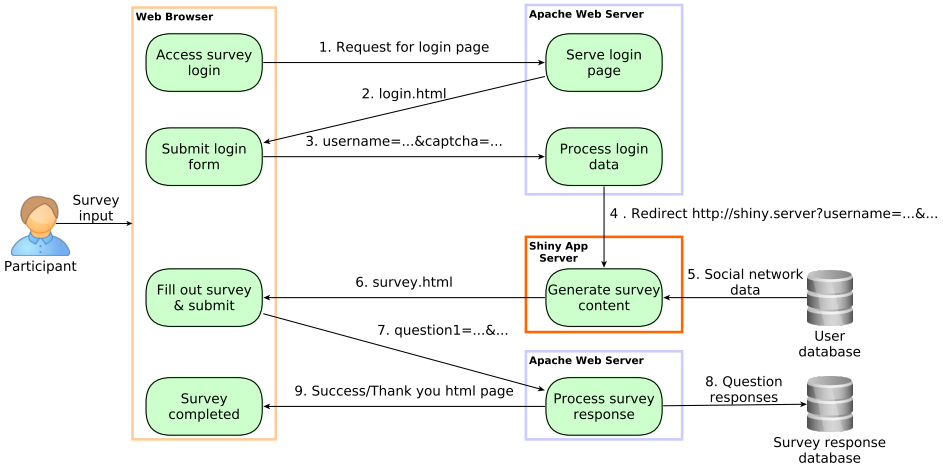


Figure 4: Event sequence and data flow for survey execution.

specific SNA results. For example, one could extend the above code to show only the vertex induced subgraph of nodes with a particularly high centrality or show only the relationships that exist between actors during specific temporal periods using sliders to select the time range of an evolving network.

Survey execution process

We now introduce the execution of the survey application, the data flow, and the general architectural elements used to accomplish the goals of each phase. The survey is broken up into three main phases discussed in detail below, namely the login phase, the survey completion phase, and the response collection phase. The login phase is used to identify the survey participant so that the appropriate visual elements can be generated for this specific user. Next, the shiny application server queries the database storing user data to generate the appropriate survey content for the specific user. Finally, the survey responses are collected and subsequently stored in a relational database so that further analysis can be easily performed. Figure 4 illustrates the sequence of events and the data exchanged between the main architectural elements. An example survey can be found at the following site: `http://rfhinf067.hs-regensburg.de/survey/?p.id=1&c.id=1`.

To design a survey, all that is necessary is to modify the shiny server R script, the PHP script for processing the responses, and the database schema that stores the survey responses. However, our application provides several generic question types and a database schema that should cover most use cases.

Login phase

The login process primarily serves to acquire the necessary user information to generate the appropriate survey content, in most cases only a user name or email address is sufficient. We also include a Captcha authentication to avoid problems associated with automated attempts to access the survey. The login page is hosted on a standard Apache web server and uses basic JavaScript to generate a URL that contains the user information as URL parameters. The login phase is shown as step 1 through 4 in Figure 4. For example, user Jane Doe accesses the login page and enters the email address j.doe@email.com, then the URL `shiny.server?username=j.doe@email.com` is generated and the user is automatically redirected to the generated shiny application after the login form has been submitted and the Captcha has been authenticated. The URL parameters allow you to create a custom survey for each individual survey participant. For example, you may wish to display the participant's ego network and ask specific question about the network's structure or authenticity with regard to capturing real-world relationships.

The login step does not contain security features that would prevent someone from logging into the survey using someone elses email. If security is a primary concern, an alternative solution is to append a unique string of characters as a URL parameter and forgo the login process entirely by instead sending an e-mail to each survey participant containing the unique link. For example, instead of generating the URL shown above from the login process, the participant would receive a link of the form `shiny.server?username=jvnoqk91m`. Then in the shiny server application, the unique string can be matched to the user to create the appropriate survey content.

```
graph <- reactive({
                  ## Parse URL for parameters into key value pairs
                  parseQueryString(clientData$url_search)})

                  ## Retrieve username parameter
                  username <- as.character(query()['username'])

                  ## Query database for subgraph containing the person
                  query.database(username)})
```

Figure 5: Example reactive expression for retrieving URL parameters.

Survey completion phase

The events of this phase are illustrated in steps 5 and 6 of Figure 4. In the survey completion phase, the user is presented with the survey content that is automatically generated based on the individual's login data. This feature allows the questions, response fields, and figures to be altered to present information that is most relevant to the particular survey participant. For example, in our survey we asked participants about whether our community detection algorithm accurately identified meaningful communities and what commonalities were responsible for producing the community. We used the login parameters to select the specific community that the survey participant was found in and displayed the specific subgraph. The URL parameters are retrieved using the code shown in Figure 5 and is located in the 'server.R' script. The example code can be extended to use any value to identify the survey participant. In the login phase description, we discuss a more secure way of performing the login by using a unique string instead of the users name and e-mail address. This code example can easily be adapted to retrieve the unique string from the URL parameters to implement the more secure login approach.

In Figure 5, the URL parameters containing the user login data are first parsed and stored as key value pairs. The user name is then retrieved and used to query the database containing the user data. Using this information we can generate the specific subgraph for each survey participant. This portion of the script can be edited to query for any user specific data. Alternatively, if your user data is stored in files, you can edit this component to read specific user files.

It was our experience that using auto-completion was very helpful for questions where the response was a user name. Since many of the mentioned names should already be in the database, we were able to take advantage of this to provide auto-completion. This allowed us to maintain consistency of names between survey response and our database. In addition, the name consistency made the processing of the responses much easier and yielded higher quality results. Shiny does not provide any auto-completion feature, however, we were able to use ShinySky to achieve this functionality.

Response collection phase

The final phase is illustrated in Figure 4 in steps 7 through 9. Upon completion and submission of the survey, the responses are sent using HTTP POST to a PHP script running on the Apache server. The PHP script checks for errors in the responses and cleans erroneous or troublesome characters. Once the responses have been cleaned and verified they are then saved to the database. Once the data have been successfully saved to the database, the user is redirected to a "thank you" page to confirm that their submission was successful. You may wish to edit the PHP script and database schema to suit your particular needs since they relate to the specific survey questions. Alternatively, you may simply write the survey responses to a file. Since R has sufficient support for MySQL, we found that saving the responses to a database made the analysis of the responses relatively simple.

Discussion and future work

In our research focused on validating social network analysis techniques for the purpose of identifying the organizational structure of open source developers (Joblin et al., 2015), we found this survey application to be extremely valuable. Our research goal was to determine if developer networks, constructed from operational data stored in the version control system, contain information that is reflective of the real world. More specifically, the survey results indicated that community-detection techniques are effective for decomposing developer networks into communities that reflect important real-world relationships. We were able to show that the developer communities are comprised of a

collection of well coordinated individuals with common goals, interests, and shared mental models. In the software engineering domain, this kind of information is incredibly valuable for identifying where coordination challenges are most likely to arise and negatively impact developer productivity or code quality (Cataldo and Herbsleb, 2013; Pinzger et al., 2008; Nagappan et al., 2008; Meneely et al., 2008), effective coordination are becoming increasingly important. With the support of R packages, applying unsupervised machine learning algorithms to networks is no longer a significant technical challenge, but the challenge of evaluating the results with respect to the ground truth model of reality still requires attention. In particular, without demonstrating that the results of applying SNA have real-world significance and validity, our research contribution would have had only a very limited impact. On this basis, we see that the challenge of validating the information that can be gleaned from social networks is fundamental to developing effective network analysis methodology that is capable of delivering valuable insights. Since the phenomenon of globally distributed collaboration is becoming common practice in many domains, we see this survey application as an important contribution to the general research community. In the future, we fully intend to reuse and improve this web-based survey application with one of our primary goals being to turn the application into an R package that is easier to setup and use.

Conclusion

Through the use of multiple R packages, we were able to successfully develop a powerful survey instrument for validating SNA approaches. Surveys for the validation of SNA techniques demand unique features that are not currently provided by existing tools. Our application provides the fundamental structural elements to conduct a large scale and fully-automated social-network based survey. Furthermore, we provide a fully-functional application that addresses the fundamental threats to validity in the social network analysis domain. Additionally, the basic application provides a substantial foundation to support the development of more elaborate survey instruments.

The SNA community is rapidly growing with many new and exciting techniques frequently being proposed, however, the real power of SNA lies within its accurate representation of the real-world. In order to achieve congruency between reality and the insights gleaned through SNA, appropriate validation tools are a necessary requirement. In this paper, we presented an example application that significantly lowers the barrier to entry for conducting social-network based surveys. By using this survey application to validate SNA techniques, one is able to more easily test the reliability and validity of SNA approaches and increase the scientific rigor of the field.

Bibliography

Y.-Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong. Analysis of topological characteristics of huge online social networking services. In *Proceedings of the International Conference on World Wide Web*, pages 835–844. ACM, 2007. [p149]

P. Barbera and M. Piccirilli. *Rfacebook: Access to Facebook API via R*, 2015. URL http://CRAN.R-project.org/package=Rfacebook. R package version 0.6. [p149]

C. T. Butts. *sna: Tools for Social Network Analysis*, 2014. URL http://CRAN.R-project.org/package=sna. R package version 2.3-2. [p149]

M. Cataldo and J. D. Herbsleb. Coordination breakdowns and their impact on development productivity and software failures. *IEEE Transactions on Software Engineering*, 39(3):343–360, 2013. [p157]

W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson. *shiny: Web Application Framework for R*, 2015. URL http://CRAN.R-project.org/package=shiny. R package version 0.12.2. [p151]

G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL http://igraph.org. [p149]

J. Donath and D. Boyd. Public displays of connection. *BT Technology Journal*, 22(4):71–82, 2004. [p149]

R. Gentleman, E. Whalen, W. Huber, and S. Falcon. *graph: A package to handle graph data structures*. R package version 1.44.1. [p149]

J. Gentry. *twitteR: R Based Twitter Client*, 2015. URL http://CRAN.R-project.org/package=twitteR. R package version 1.1.9. [p149]

M. Joblin, W. Mauerer, S. Apel, J. Siegmund, and D. Riehle. From developer networks to verified communities: A fine-grained approach. In *Proceedings of the International Conference on Software Engineering*, 2015. [p149, 156]

R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. In *Link Mining: Models, Algorithms, and Applications*, pages 337–357. Springer, 2010. [p149]

A. Meneely, L. Williams, W. Snipes, and J. Osborne. Predicting failures with developer networks and social network analysis. In *Proceedings of the Foundations of Software Engineering*, pages 13–23. ACM, 2008. [p157]

A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the Conference on Internet Measurement*, pages 29–42. ACM, 2007. [p149]

N. Nagappan, B. Murphy, and V. Basili. The influence of organizational structure on software quality: An empirical case study. In *Proceedings of the International Conference on Software Engineering*, pages 521–530. ACM, 2008. [p157]

M. Pinzger, N. Nagappan, and B. Murphy. Can developer-module networks predict failures? In *Proceedings of the International Symposium on Foundations of Software Engineering*, pages 2–12. ACM, 2008. [p157]

C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao. User interactions in social networks and their implications. In *Proceedings of the European Conference on Computer Systems*, pages 205–218. ACM, 2009. [p149]

Mitchell Joblin
Siemens AG
Wladimirstraße 3
91058 Erlangen
Germany
mitchell.joblin.ext@siemens.com

Wolfgang Mauerer
Siemens AG
Technical University of Applied Science Regensburg
Universitätsstraße 31
93058 Regensburg
Germany
wolfgang.mauerer@oth-regensburg.de

Exploring Interaction Effects in Two-Factor Studies using the hiddenf Package in R.

by Christopher T. Franck and Jason A. Osborne

Abstract In crossed, two-factor studies with one observation per factor-level combination, interaction effects between factors can be hard to detect and can make the choice of a suitable statistical model difficult. This article describes **hiddenf**, an R package that enables users to quantify and characterize a certain form of interaction in two-factor layouts. When effects of one factor (a) fall into two groups depending on the level of another factor, and (b) are constant within these groups, the interaction pattern is deemed "hidden additivity" because within groups, the effects of the two factors are additive, while between groups the factors are allowed to interact. The **hiddenf** software can be used to estimate, test, and report an appropriate factorial effects model corresponding to hidden additivity, which is intermediate between the unavailable full factorial model and the overly-simplistic additive model. Further, the software also conducts five statistical tests for interaction proposed between 1949 and 2014. A collection of 17 datasets is used for illustration.

Introduction

The crossed, two-factor layout with one observation per factor-level combination is a simple and widely used plan. These designs arise in the context of (a) completely randomized two-factor experiments, (b) randomized complete block experiments that block on a source of nuisance variability, and (c) observational studies with two factors. If the effects of the two factors do not interact, then this design permits inference for both factors simultaneously. The following model is commonly used for data collected using these designs:

$$y_{ij} = \mu + \tau_i + \beta_j + \epsilon_{ij}, \tag{1}$$

where $i = 1, \ldots, c$ and $j = 1, \ldots, r$ are indices for levels of the two factors C and R, μ represents the overall mean of the outcome y_{ij}, and the τ_i and β_j terms quantify the effects of factors C and R. The errors are frequently assumed independent and identically normally distributed, with constant error variance σ^2. Since data from these designs are easy to display using two-way tables, frequently the levels of factors R and C are called "rows" and "columns," respectively. In this report (and the **hiddenf** manual), Factor R is referred to as the "row" or "grouped" factor, while factor C is called the "column" factor.

Model (1) assumes that the effect of the row factor on the response y_{ij} does not depend on the level of the column factor, and vice versa. In statistical parlance this is known as the assumption of "additivity." This assumption is sometimes made out of necessity as the alternative of including the interaction term from the full factorial effects model precludes statistical inference. After computing the interaction mean square, zero degrees of freedom remain with which to estimate the error variance, σ^2. Data arising from (1) are shown in Figure 1 panel A.

If the effect of factor C varies across levels of the factor R, then there is statistical interaction. In this case (1) is misspecified. Graphical assessment of additivity is commonly made using an interaction plot. Figure 1 panel B is an interaction plot for the `cjejuni.mtx` data set included in the **hiddenf** package (Qiu, 2013).

Since the data in Figure 1 panel B do not exhibit parallelism, there is graphical evidence to suggest that (1) may not be a suitable model. This is problematic because inferences conducted under (1) will be incorrect if (1) fails to describe the true nature of the variables under investigation. Fortunately, many methods have been devised to assess non-additivity in these designs, including those in Tukey (1949), Anscombe and Tukey (1963), Mandel (1961), Mandel (1971), Johnson and Graybill (1972), Hirotsu (1982), Kharrati-Kopaei and Sadooghi-Alvandi (2007), Tusell (1990), Boik (1993a), Franck et al. (2013), and Malik et al. (2014). Alin and Kurt (2006) provide a review of methods available up to 2006. We return to the analysis of these and other data in subsequent sections of the paper.

Despite the wide usage of unreplicated two-way designs, computational tools to assess non-additivity

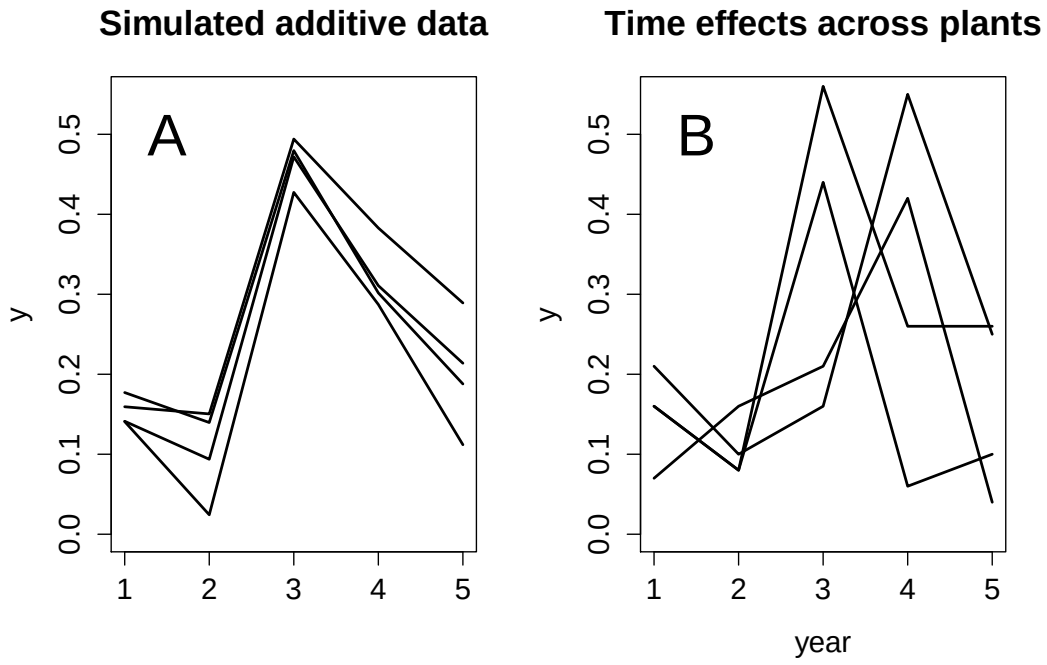


Figure 1: Interaction plots. Lines correspond to levels of factor *R* and tick marks on the horizontal axis correspond to levels of factor *C*. Panel A is simulated data from (1). These data do not exhibit statistical interaction so departure from parallel lines is due to random noise. Panel B is `cjejuni.mtx` data, which visually exhibit non-additivity.

are lacking, particularly for more recent methods. Currently, the **additivityTests** package (Simecek and Simeckova, 2012) provides test statistics, critical thresholds, and binary reject/fail to reject decisions for tests proposed in Tukey (1949), Mandel (1961), Boik (1993a), Tusell (1990), Johnson and Graybill (1972), and Simecek and Simeckova (2012). The **additivityTests** package does not produce p-values.

The hiddenf package

This paper describes the R package **hiddenf**, which makes available several tools for the analysis of non-additivity in unreplicated two-way layouts. This package contributes to available computational resources by (a) providing statistical and graphical diagnostic tools within the context of hidden additivity that enable the user to go beyond overall tests of additivity towards the inferential goal of characterizing and quantifying the magnitude of interaction, and (b) providing p-value computations for five methods to detect non-additivity. Supported tests include those proposed in Tukey (1949), Mandel (1961), Kharrati-Kopaei and Sadooghi-Alvandi (2007), Franck et al. (2013), and Malik et al. (2014). The latter three are newly available in an open-source repository via the **hiddenf** package, which is available from the Comprehensive R Archive Network (CRAN).

Hidden additivity occurs when the levels of factor *R* fall into a smaller number of groups, such that within these groups the effect of factor *C* is constant across levels of factor *R*, but there is $C \times group$ interaction (Franck et al., 2013). Group membership of levels of *R* can be regarded as latent, unobservable random variables. Inspection of Figure 1 panel B provides an example. If one group is formed from the two lines (processing plants) that reach their peak in year 3, and another from the two lines that reach their peak in year 4, then the year and plant effects are roughly additive within these groups.

The `HiddenF()` function accepts an $r \times c$ data matrix as input and returns an object of the *HiddenF* class. Objects of this class store the p-value from the all-configurations maximum interaction F (ACMIF) test (Franck et al., 2013) for hidden additivity, the number of configurations under consideration, the configuration that achieves the maximum interaction F score (or maximum hidden additivity), and the data as a list. Several generic functions, including `print()`, `anova()`, `summary()` and `plot()`, can be applied to objects of the *HiddenF* class to produce output useful for the quantification and characterization of interaction. By default, the `HiddenF()` function considers the row factor as the

variable to group. Submitting a transposed matrix executes the grouping on the column variable. Deciding which factor to group is subjective, and can be approached using domain knowledge or a trial-and-error approach. Figure 2 summarizes the functionality of the **hiddenf** package.

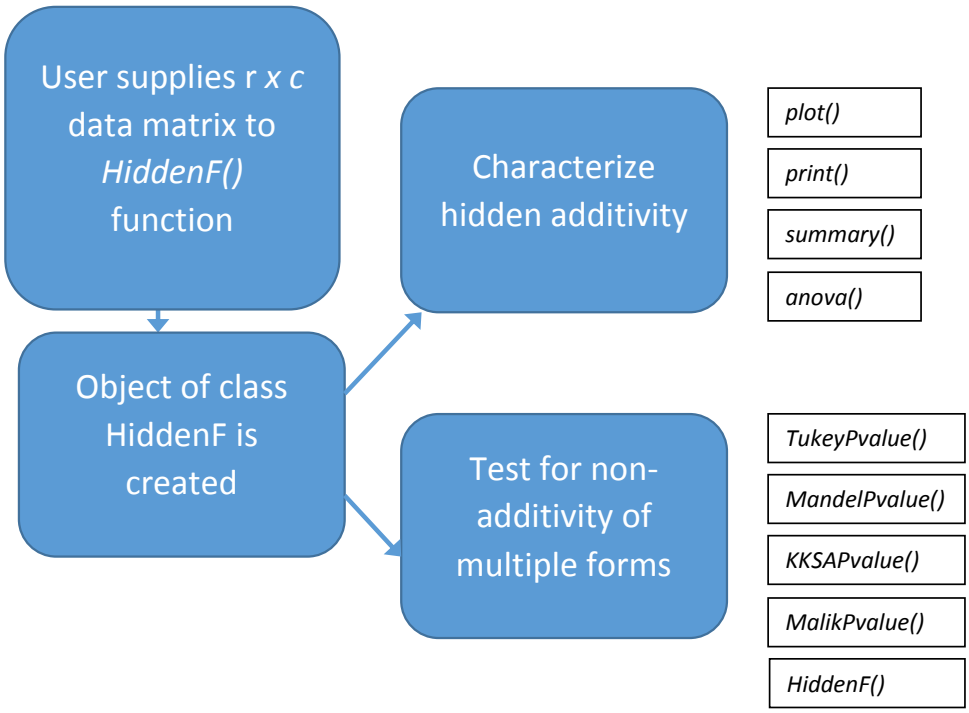


Figure 2: Flowchart of **hiddenf** functionality. The user supplies a $r \times c$ data matrix to the `HiddenF()` function, which returns an object of the *HiddenF* class. The user can then characterize hidden additivity and obtain p-values for tests of non-additivity.

The remainder of the paper is organized as follows. The *C. jejuni* data are first described as a relevant example. The following section reviews the testing procedure for hidden additivity. Functionality to detect hidden additivity using **hiddenf** is then detailed. Four other supported methods to detect non-additivity are then reviewed. Seventeen data sets are described and explored using the methods supported by **hiddenf**. The article concludes with a summary.

Example

We now return to the data from Figure 1 panel B. The vertical axis in this plot is the proportion of disease-resistant bacteria samples of the *C. jejuni* strain taken from turkey processing facilities. These data were collected over a five year period across four processing plants in North Carolina. Each line corresponds to a plant, and the year labels correspond to 2008-2012. The matrix `cjejuni.mtx` contains the fractions of bacteria that are classified as the strain *C. jejuni*.

```
> library(hiddenf)
> data(cjejuni.mtx)
> cjejuni.mtx

     [,1] [,2] [,3] [,4] [,5]
[1,] 0.16 0.08 0.44 0.06 0.10
[2,] 0.21 0.10 0.16 0.55 0.25
[3,] 0.16 0.08 0.56 0.26 0.26
[4,] 0.07 0.16 0.21 0.42 0.04
```

Testing for hidden additivity

Hidden additivity is a useful concept for the analysis of many types of data because the structure is easy to visualize and interpret. The ACMIF test (Franck et al., 2013) to detect hidden additivity assigns classical significance testing-based p-values. A combined testing and plotting approach allows the researcher to interpret the way in which column effects change across groups of the rows.

The ACMIF test works by (1) considering each placement of factor R levels into two nonempty groups, (2) testing factor $C \times group$ interaction, and (3) considering the test that provides the most evidence of interaction (and greatest additivity of C and R within groups) after correcting for multiplicity using a Bonferroni adjustment. Simulation suggests that the Bonferroni correction is not overly conservative for $r \leq 7$ despite the magnitude of the correction factor $2^{r-1} - 1$, because the Bonferroni adjusted thresholds are remarkably similar to simulated critical thresholds for a variety of c and r values (Franck et al., 2013). If g is an index for the latent group membership such that $g = 1, 2$, then the factorial effects model corresponding to the hidden additivity structure is given by

$$y_{ij}^{g} = \mu + \tau_i + \gamma_g + (\tau\gamma)_{ig} + \beta_{j(g)} + \epsilon_{ijg} \tag{2}$$

where μ is the overall mean, τ_i and γ_g are the main effects of factor C and the grouping variable, $(\tau\gamma)_{ig}$ represents factor $C \times group$ interaction, and $\beta_{j(g)}$ is the effect of factor R, nested within group. The ACMIF test is based on the largest F-ratio corresponding to $H_0 : (\tau\gamma)_{ig} = 0$ among all assignments of R factor levels into two non-empty groups.

Characterizing hidden additivity

Objects in the *HiddenF* class can be used to characterize hidden additivity further with application of the generic `plot()`, `print()`, `summary()`, and `anova()` functions. The following code produces an enhanced interaction plot to help visualize hidden additivity (Figure 3):

```
> cjejuni.out <-HiddenF(cjejuni.mtx)
> plot(cjejuni.out, main="Hidden Additivity of time effects across plants",
+ rfactor="plant", cfactor="year", colorvec=c("blue", "red"), lwd=3, legendx=TRUE)
```

The enhanced interaction plot in Figure 3 displays levels of factor C (year) on the horizontal axis, levels of factor R (plant) using lines that are visually distinguished by line type, and the outcome values on the vertical axis. Color is used to display which levels of factor R are assigned to which groups based on the maximal interaction F statistic among all possible configurations and (2). Note that the grouping colors appear whether or not the ACMIF p-value (test for $C \times group$) to detect hidden additivity is significant. The default colors are black and red for the two groups, but they can be customized by supplying an argument called `colorvec` which is a vector of length two giving color names. Another optional argument, `legendx`, may be set to `TRUE` in order to provide a legend whose location will be determined according to where on the plot the user clicks.

Because of the ellipsis(...), arguments such as `main` (for a title) or `lwd` (for specifying line widths) can be passed to `matplot` or `legend`. The arguments `lty`, `type`, `ylab`, and `xlab` are utilized by the `plot.HiddenF()` function to produce the enhanced interaction plot and therefore are not available for customization by the user.

The `print()` function returns results from the ACMIF test.

```
> print(cjejuni.out)
The ACMIF test for the hidden additivity form of interaction
F=8.965 p-value =0.03309 df=4,8
(Bonferroni-adjusted for all 7 possible configurations)
```

This output can also be obtained by typing the name of an object of class *HiddenF* into the console. The p-value above has been adjusted for multiplicity using the Bonferroni multiplier of $2^{4-1} - 1 = 7$. Additionally, the `method` argument in the `print.HiddenF()` function supports all of the methods supported by the **hiddenf** package discussed later. To see which of these seven possible configurations of rows in two non-empty groups leads to the greatest additivity within groups, use the generic `summary()` function, which returns information useful for analysis based on (2):

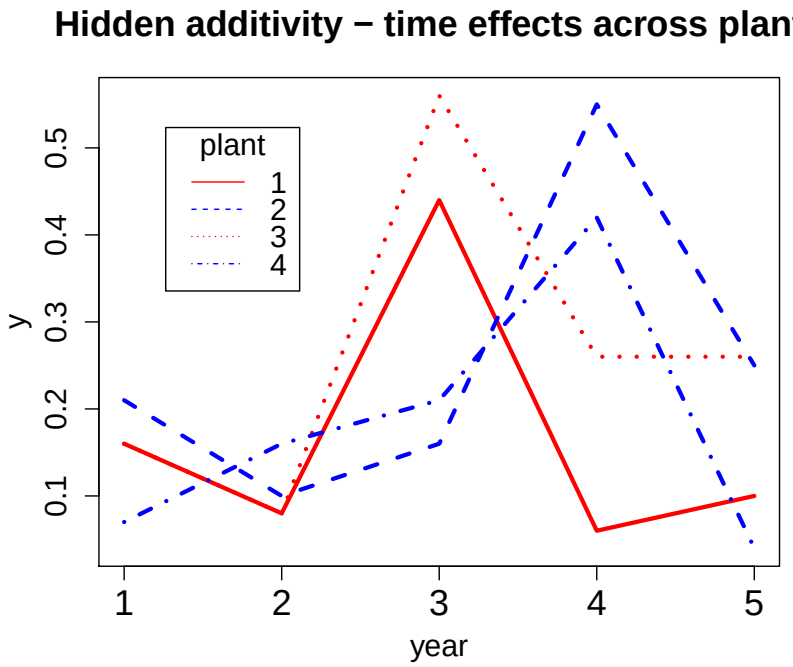


Figure 3: Hidden additivity plot for the `cjejuni.mtx` data. Lines are color-coded to correspond to the configuration that achieves the maximal $C \times group$ interaction F statistic. Individual rows are distinguished by line type to allow the user to easily identify specific contributions of rows to hidden additivity .

```
> summary(cjejuni.out)
Number of configurations: 7
Minimum adjusted pvalue: 0.03308869

Rows in group 1: 1 3
Rows in group 2: 2 4

Column means for grp 1: 0.16 0.08 0.5 0.16 0.18
Column means for grp 2: 0.14 0.13 0.185 0.485 0.145
```

The output above includes the number of configurations under consideration, the smallest Bonferroni adjusted p-value among these, the identity of the rows that fall into each group and the means across levels of C for both groups. Each of these items is returned in a list (not shown). To produce an ANOVA table:

```
> anova(cjejuni.out)
The ACMIF test for the hidden additivity form of interaction
Analysis of Variance Table

Response: y
           Df  Sum Sq  Mean Sq F value   Pr(>F)
group       1 0.00000 0.000005  0.0009 0.977350
col         4 0.18753 0.046882  8.0450 0.006606
row         2 0.03673 0.018365  3.1514 0.097874
group:col   4 0.20897 0.052243  8.9648 0.004727
Residuals   8 0.04662 0.005828
C.Total    19 0.47986
(Pvalues in ANOVA table are NOT corrected for multiplicity.)
```

For these data, the interaction between time (`col`) and plant group (`group`) accounts for more variability (43%) than any other term in the hidden additivity model. This partial coefficient of determination is computed by dividing the group $\times$ column sums of squares by the total sum of squares displayed above. The anova() function can also accommodate other tests for non-additivity by specifying the `method` argument as "KKSA", "Mandel" or "Tukey" (see Other Tests for Non-additivity section).

Levels of factor *C* can be grouped instead by transposing the input data matrix before applying the `HiddenF()` function. The output below suggests no statistical evidence for hidden additivity in the levels of factor *C*.

```
> cjejuni.trans.mtx<-t(cjejuni.mtx)
> cjejuni.trans.out<-HiddenF(cjejuni.trans.mtx)
> print(cjejuni.trans.out)
The ACMIF test for the hidden additivity form of interaction
F=3.63 p-value =0.8671 df=3,9
(Bonferroni-adjusted for all 15 possible configurations)
```

Note that support for more than 20 rows is not yet included due to the exponential increase in computational demand as the number of grouped levels increases. Factors with more levels than this will be accommodated in future versions of the software.

Centering

The `center` option in the plot command allows the user to graph the hidden additivity plot with data centered at the row means to better see the concordance of rows with regard to column effects. This is a way of de-noising the plot. Figure 4 demonstrates this feature using data from Graybill (1954). These data will be further analyzed later. Inspection of the right panel of Figure 4 suggests that the third genotype appears to give inferior yields for rows in the red group (relative to the performance of other genotypes in these blocks), but does better for rows colored black. To produce this plot:

```
> data(Graybill.mtx)
> Graybill.out <- HiddenF(Graybill.mtx)
> par(mfrow=c(1,2))
> plot(Graybill.out)
> plot(Graybill.out, center=TRUE, main="Hidden Additivity plot\ncenter=TRUE")
```

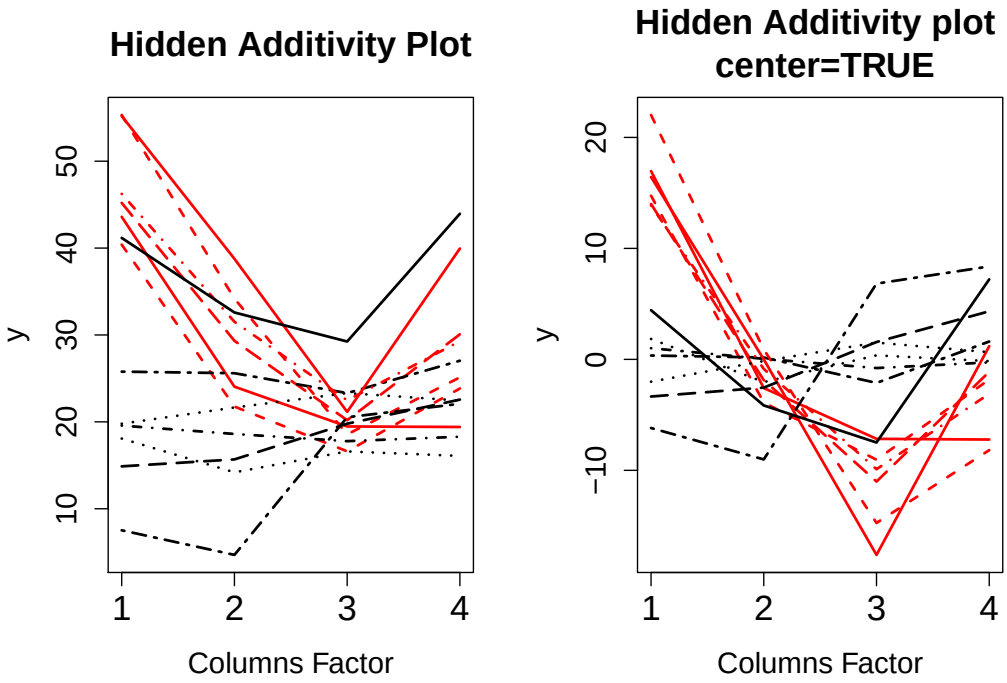


Figure 4: Hidden additivity plots for the wheat yield data (Graybill, 1954). The left and right panels exhibit the uncentered and centered graphical options, respectively.

Other tests for non-additivity

The ACMIF approach Franck et al. (2013) to detect hidden additivity has been described above. The subsections below include code examples and briefly review the other four supported approaches to

detect non-additivity in unreplicated studies.

One degree of freedom approach

The one degree of freedom test (Tukey, 1949) is the earliest and most widely-known approach. The following model for non-additivity corresponds to Tukey's procedure:

$$y_{ij} = \mu + \tau_i + \beta_j + \nu\tau_i\beta_j + \epsilon_{ij}. \tag{3}$$

A hypothesis test for this approach is based on the null hypothesis $H_0 : \nu = 0$. Under the null, (3) reduces to (1). The one degree of freedom test arises by fitting the squared fitted values from the additive model as a model term, then assessing the partial F-test for statistical significance corresponding to this term.

The `TukeyPvalue()` function accepts an object of class *HiddenF* and returns a list that contains the p-value for the one degree of freedom test and an *lm* object that contains the model that includes the squared predictions from (1) as described above. The `method` argument in the `anova.HiddenF()` function can be used to produce an ANOVA table for this test. Since the p-value is large, the output below does not provide statistical evidence for non-additivity of the form suggested in (3).

```
> TukeyPvalue(cjejuni.out)
$pvalue
[1] 0.7077391

$singledf.out

Call:
lm(formula = y ~ rows + cols + psq, data = hfobj$tall)

Coefficients:
(Intercept)        rows2        rows3        rows4        cols2        cols3
   0.095454     0.029036     0.030905     0.005445    -0.026989     0.043692
      cols4        cols5          psq
   0.044568    0.006369     1.569601
```

Rows-linear approach

An extension of the one degree of freedom test is based on the following model (Mandel, 1961):

$$y_{ij} = \mu + \tau_i + \beta_j + \theta_i\beta_j + \epsilon_{ij}, \tag{4}$$

where ϵ_{ij} are i.i.d. $N(0, \sigma^2)$. The test for rows-linear non-additivity is based on the null hypothesis $H_0 : \theta_i = 0$ for $i = 1, \ldots, t$. Under this null, (4) reduces to (1).

By setting $\mu_i = \mu + \tau_i$ and $\theta_i = (\phi_i - 1)$, the response variable y_{ij} in (4) can be represented as an intercept μ_i and slope ϕ_i that depend on effect β_j. Letting i and j index "rows" and "columns" respectively leads to the phrase "rows-linear" model, which was established later (see Alin and Kurt, 2006).

The ANOVA table sum of squares associated with the θ_i term is

$$SS_{rowlin} = \sum_i \left(\frac{\sum_j y_{ij}(\bar{y}_{.j} - \bar{y}_{..})}{\sum_j (\bar{y}_{.j} - \bar{y}_{..})^2} - 1\right)^2 \sum_j (\bar{y}_{.j} - \bar{y}_{..})^2. \tag{5}$$

The residual sum of squares has the following form:

$$SS_{res} = \sum_i \sum_j \left[(y_{ij} - \bar{y}_{i.}) - \frac{\sum_j y_{ij}(\bar{y}_{.j} - \bar{y}_{..})}{\sum_j (\bar{y}_{.j} - \bar{y}_{..})^2}(\bar{y}_{.j} - \bar{y}_{..})\right]^2. \tag{6}$$

The test statistic is:

$$F_{rowlin} = \frac{SS_{rowlin}/(a-1)}{SS_{res}/(a-1)(b-2)}. \tag{7}$$

The statistic F_{rowlin} has an F distribution with $(a-1)$ numerator and $(a-1)(b-2)$ denominator degrees of freedom under the null hypothesis. Construction of a columns-linear test is accomplished by defining a *HiddenF* object on the transposed data matrix and calling the `MandelPvalue()` in a similar fashion.

The `MandelPvalue()` function accepts an object of class *HiddenF* and returns a list that includes a p-value, sums of squares, F ratio, and degrees of freedom for the rows-linear test. The output below fails to detect interaction using Mandel's rows-linear test:

```
> MandelPvalue(cjejuni.out)
$pvalue
[1] 0.9458807

$SumSq
      SSRow       SSCol    SSMandel         SSE       SSTot
0.036735000 0.187530000 0.009849526 0.245740474 0.479855000

$Fratio
[1] 0.120243

$df
[1] 3 9
```

Error mean square subtable comparison approach

The error mean square (MSE) subtable comparison approach (Kharrati-Kopaei and Sadooghi-Alvandi, 2007) forms an F statistic that is the ratio of residual sums of squares that arise from rows being placed into two groups, with each group containing at least two rows,

$$F_{KKSA} = \frac{(r_2-1)SS_{E1}}{(r_1-1)SS_{E2}}. \tag{8}$$

Under the assumptions of additivity and homogeneous variance, F_{KKSA} has an F distribution with $(r_1-1)(c-1)$ numerator and $(r_2-1)(c-1)$ denominator degrees of freedom under the null hypothesis, no matter which rows are placed into which group. This approach suggests non-additivity when error mean square values within the subtables are discrepant.

As with the ACMIF procedure, grouping is subjective and is accomplished by choosing one of the two factors upon which to form the groups, and then placing that factor's levels into two groups. Different groupings result in different p-values. Groups are typically chosen based on *a priori* knowledge, inspection of an interaction plot, or by screening several candidate groupings and applying multiplicity adjustment to the resulting p-values. The **hiddenf** package adopts the authors' (Kharrati-Kopaei and Sadooghi-Alvandi, 2007) suggestion of assessing the $2^{r-1}-r-1$ possible groupings for factor R and Bonferroni adjusting the resulting p-values to achieve an α level test.

The `KKSAPvalue()` function accepts an object of class *HiddenF* and returns a list that contains the maximal F statistic among configurations, a Bonferroni adjusted p-value, a length r vector that indicates the groupings of levels of factor R that correspond to the calculated p-value, and numerator and denominator degrees of freedom for the test statistic. Data may be transposed in order to form subtables according to factor C. At significance level $\alpha = .05$ the output below does not reject the null hypothesis of additivity according to the MSE subtable comparison approach when grouping levels of factor R.

```
> t(KKSAPvalue(cjejuni.out))
         fmax     pvalue  grp.vector   NumDf DenomDf
[1,]   1.748821     1     Numeric,4      4     4
```

Residual clustering approach

A recent approach (Malik et al., 2014) considers non-additivity elicited due to cell values which are remote relative to the additive model. Not all cells are assumed to contribute to the interaction pattern. The method detects non-additivity by exploiting the large residuals that arise under this structure.

Execution of the residual clustering method proceeds as follows. First, residuals from (1) $\hat{r}_{ij(1)} = y_{ij} - \hat{y}_{ij(1)}$ are placed into 3 groups using k-means clustering (MacQueen, 1967) in one dimension. Then, a two degrees of freedom cluster effect is incorporated into the additive model:

$$y_{ij} = \mu + \tau_i + \beta_j + \kappa_{k(ij)} + \epsilon_{ij} \tag{9}$$

where $\kappa_{k(ij)}$ represents the effect of the kth cluster, $k = 1, 2, 3$. The subscript $k(ij)$ indicates that the $ijth$ residual is assigned to exactly one of the k clusters for $i = 1, \ldots, c$ and $j = 1, \ldots, r$. Since the cluster term is suggested by the data, the partial F test corresponding to the cluster term does *not* exhibit a central F distribution under the null. Rather, the null distribution for this statistic may be approximated via Monte Carlo simulation for the purpose of conducting statistical inference. Critical thresholds for a variety of c and r are available (Malik et al., 2014). The **hiddenf** package computes p-values and critical thresholds for user-supplied c and r using the functions `MalikPvalue` and `MalikTab`, respectively. By default $N = 500$ Monte Carlo replicates are used to approximate the p-value, which guarantees the standard error of the p-value $SE(p - value) \leq 0.023$.

The `MalikPvalue` function accepts an object of class *HiddenF* and returns a Monte Carlo p-value, observed test statistic, and Monte Carlo sample size used in the computation. The k-means approach uses the method of Hartigan and Wong (1979) with 100 starting values per Monte Carlo iteration. The output below suggests that the *C. jejuni* data do not exhibit non-additivity of the form suggested by this method. This approach is invariant to data transposition.

```
> t(MalikPvalue(cjejuni.out, N=10000))
(Pvalue from Malik's test estimated with N=10000 Monte Carlo datasets)
     pvalue    Tc      N
[1,] 0.8498 40.97983 10000
```

The *MalikTab* function accepts as input a user-specified r and c, and returns a Monte Carlo estimate of the 90%, 95%, and 99% for the null distribution of the test statistic. A variety of such thresholds are provided in Malik et al. (2014). The default number of Monte Carlo replicates is $N = 1,000$. An example is given below.

```
> Mtab.24.6<-MalikTab(r=24, c=6, N=10000)
> ls(Mtab.24.6)
[1] "q"     "Tcsim"
> Mtab.24.6\$q
      r        c      99%      95%      90%
 24.0000   6.0000 445.5725 404.3399 384.0591
```

The `additivityPvalues` function

The `additivityPvalues` function accepts an object of class *HiddenF* and returns p-values for each of the supported methods.

```
> t(additivityPvalues(cjejuni.out))
(Pvalue from Malik's test estimated with N=500 Monte Carlo datasets)
     Malik.pvalue Mandel.pvalue Tukey.pvalue KKSA.pvalue ACMIF.pvalue
[1,]    0.834        0.9459       0.7077           1       0.0331
```

Data examples

Table 1 summarizes an investigation of 17 data sets using the methods supported by **hiddenf**. These examples are taken from bioinformatic, agricultural, industrial, and medical settings, and are all publicly available. To our knowledge, this is the largest collection and description of data that has been used to study non-additivity in the literature to date. Table 1 includes columns for references, labels, and p-values. Note that ACMIF c, columns-linear, and KKSA c are obtained by submitting the transposed data matrix to `HiddenF()`. A brief description of each data set follows.

The "Liming" data arises from an experiment that explores the utility of seven types of blast furnace slags as liming material in agriculture on three types of soil (Carter et al., 1951). The data were analyzed in the context of non-additivity in Johnson and Graybill (1972) and Kharrati-Kopaei and Sadooghi-Alvandi (2007). The outcome variable is yields of corn in bushels per acre.

The "Penicillin" data set (Davies and Goldsmith, 1972) assesses variability between six samples of penicillin on 24 plates. The outcome variable is the diameter of the zone of inhibition. The data were analyzed for non-additivity in Malik et al. (2014).

The "Osmotic" data set (Kharrati-Kopaei and Sadooghi-Alvandi, 2007) explores five varieties of safflower grown in solutions with six osmotic potentials. Safflowers are an important crop due to their oil which can be used for cooking and their flowers which can be used as a substitute for saffron. The outcome variable is average root weight.

The "Fertilizer" example originally appeared in Ostle (1963) and was re-analyzed to detect non-additivity in Kharrati-Kopaei and Sadooghi-Alvandi (2007). This example explores the ratio of dry to wet wheat in four blocks for four fertilizers.

The "Bottles" example was originally analyzed in Ott and Snee (1973) and also explored in Boik (1993b) and Kharrati-Kopaei and Sadooghi-Alvandi (2007). The study assesses the performance of a six-headed machine on five occasions.

The "Grain Yields" data measures the yield of a grain crop in bushels per acre for five levels of fertilizer on five blocks. The data come from Ostle (1963) and are re-analyzed in Kharrati-Kopaei and Sadooghi-Alvandi (2007).

The "Absorbance" data (Mandel, 1991) measures absorbancy of wood pulp of nine polysachharide concentrations from seven different laboratories. The data are also analyzed in Alin and Kurt (2006).

The "Permeability" example examines permeability of three sheets of building material on nine different days. The data appear in Hald (1952) and Giesbrecht and Gumpertz (2004).

The "Wool" data (Lentner and Bishop, 1993) examines four cleaning processes for wool from five different batches. The outcome is losses in weight in mg of the sample after cleaning.

The "Red Blood Cells" data measures the number of red blood cells counted by five doctors in ten counting chambers. The data appear in Biggs and Macmillan (1948) and were also analyzed in Boik (1993a).

The "Tukey" data set is an illustrative example that includes three rows and four columns from Tukey (1949).

The "Ethyl Alcohol" data (Osborne et al., 1913) measures the density of aqueous solutions of ethyl alcohol at six concentrations and seven temperatures. These data were also analyzed in Mandel (1971).

The "Insecticide" data (Ott and Longnecker, 2001) comes from a complete block design that measures the impact of four plots and three insecticides on the number of string bean seedlings that emerge.

The "Rubber" data (Mandel, 1961) contains data on stress measurements in Kg/cm^2 on seven types of natural rubber vulcanizates from 11 laboratories.

The "*C. jejuni*" data measures the fraction of bacteria found on disease-resistant turkeys (Qiu, 2013). These data were collected over a five year period across four turkey plants in North Carolina. These data are displayed in Figures 1 and 3.

The "Wheat" data (Graybill, 1954) measures wheat yields in bushels per acre in four varieties in 13 locations. These data are plotted in Figure 4.

The eight Copy number variation data sets "CNV1-CNV8" compare discrepancies in the copy number signal between normal and tumor tissue samples from a comparative genomic hybridization array. Each data set corresponds to a separate location in the genome that is tested with the assay. Six dogs each had two tissue samples (one normal and one tumor) upon which the assay was conducted. The eight specific sets were selected from 5899 sets that were analyzed in a previous study and because they showed evidence of hidden additivity (Franck et al., 2013). The hidden additivity exhibited in these copy number responses might suggest the existence of multiple tumor sub-types for lymphoma in

dogs. The full data for this example may be accessed here: http://www.sciencedirect.com/science/article/pii/S0167947313001618.

The results of non-additivity tests using methods supported by the **hiddenf** package are reported in Table 1. The data come from 17 sources, and there are 24 total sets with the copy number variation contributing eight individual sets. This collection is not intended as a representative sample of the larger class of all unreplicated two-factor data, since many sets were included due to their apparent non-additivity. This exercise is intended to show the breadth of applications for which the study of non-additivity is important. To facilitate discussion, the standard p-value less than $\alpha = 0.05$ criterion is used to declare statistical significance without further multiplicity correction, although the reader is invited to interpret the raw p-values however they like. The ACMIF test for hidden additivity was significant for 16 of 24 sets when grouping levels of factor R. The test was significant for eight of 16 sets when grouping levels of factor C. The Tukey test was significant for eight of 24 sets. The rows-linear and columns-linear tests showed significant non-additivity for seven and six sets (out of 16 possible), respectively. The KKSA test grouping levels of R and C revealed non-additivity for 13 of 22 and five of 13 sets, respectively. The Malik approach yielded significant non-additivity for nine of 24 sets. Since all tests assume different restrictions on the form of interaction, no single test is optimal for every conceivable pattern. Hence, differential performance among methods is expected for different types of data. Each test excels at detecting different patterns of non-additivity.

Summary

This paper describes the main functionality of the **hiddenf** package. **hiddenf** provides descriptive and inferential tools to visualize and characterize hidden additivity. Further, **hiddenf** includes the ability to compute p-values for five tests for non-additivity. The package is illustrated using seventeen data sets spanning studies in industrial applications, agriculture, and the medical sciences. Non-additivity is evident in many of these data sets, motivating the importance of statistical interaction in unreplicated studies across a variety of scientific domains. The **hiddenf** package contributes to existing software resources specifically by making three recent tests for non-additivity available in an open-source repository for the first time (in addition to two historical methods) and by providing visualization and descriptive tools to further explore hidden additivity.

Citation	Label	r	c	ACMIF r	ACMIF c	Tukey	row-linear	col-linear	KKSA r	KKSA c	Malik
Carter et al. (1951)	Liming	7	3	0.1993	0.0110	0.9106	0.9045	0.8604	0.0068	‡	0.2471
Davies and Goldsmith (1972)	Penicillin	6	24	0.0387	◇	0.5382	0.9374	◇	1.000	◇	0.2231
Kharrati-Kopaei and Sadooghi-Alvandi (2007)	Osmotic Bars	6	5	0.2075	0.0388	0.4450	0.5300	0.1848	0.1859	0.4155	0.7453
Ostle (1963)	Fertilizer	4	4	0.0107	0.0043	0.0012	0.0146	0.0077	0.0339	0.0493	0.4373
Ott and Snee (1973)	Bottles	6	5	0.0001	0.0031	0.0003	0.0025	0.0006	0.0178	0.4925	0.8839
Ostle (1963)	Grain yields	5	5	0.6556	0.1485	0.1139	0.0103	0.4425	0.0528	1.0000	0.0403
Mandel (1991)	Absorbance	7	9	0.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	0.9972
Hald (1952)	Permeability	9	3	0.4066	0.2629	0.7844	0.7761	0.9544	0.3908	‡	0.8674
Lentner and Bishop (1993)	Wool	4	5	0.2355	0.4087	0.0359	0.0519	0.0410	0.7155	0.1998	0.4182
Biggs and Macmillan (1948)	Red blood cells	5	10	0.3291	0.0825	0.2322	0.6790	0.4903	0.0668	1.0000	0.8702
Tukey (1949)	Tukey	3	4	0.2213	0.5773	0.8566	0.8743	0.9298	‡	0.5806	0.4769
Osborne et al. (1913)	Ethyl Alcohol	6	7	<.0001	<.0001	<.0001	<.0001	<.0001	0.0002	0.0469	0.9954
Ott and Longnecker (2001)	Insecticide	3	4	0.4038	0.5569	0.6875	0.3846	0.8125	‡	0.3000	0.7603
Mandel (1961)	Rubber	11	7	<.0001	<.0001	0.3465	<.0001	0.5403	0.0021	0.0058	0.9787
Qiu (2013)	*C. jejuni*	4	5	0.0331	0.8671	0.7077	0.9459	0.8842	1.0000	0.2862	0.8443
Graybill (1954)	Wheat	13	4	<.0001	0.0070	<.0001	0.0003	<.0001	0.0320	0.0143	0.1335
Franck et al. (2013)	CNV1	6	2	0.0007	‡	0.0138	‡	‡	0.0310	‡	0.0009
	CNV2	6	2	0.0005	‡	0.0659	‡	‡	0.0010	‡	0.0164
	CNV3	6	2	<.0001	‡	0.4674	‡	‡	0.0017	‡	0.0018
	CNV4	6	2	0.0005	‡	0.1659	‡	‡	0.1249	‡	0.0182
	CNV5	6	2	0.0005	‡	0.2388	‡	‡	0.0259	‡	0.0173
	CNV6	6	2	0.0007	‡	0.6187	‡	‡	0.2799	‡	0.0224
	CNV7	6	2	0.0006	‡	0.0277	‡	‡	0.0939	‡	0.0193
	CNV8	6	2	0.0007	‡	0.1391	‡	‡	0.0340	‡	0.0216

Table 1: P-values for various tests of non-additivity supported by **hiddenf**. ‡: r and/or c insufficient for analysis. ◇: grouping on $c > 20$ for Penicillin not currently available in **hiddenf**. The Malik p-values employ N=100,000 Monte Carlo replicates.

ISSN 2073-4859

Acknowledgements

We'd like to acknowledge Zahra Shenavari of Shiraz University for helpful comments regarding the error mean square subtable comparison approach.

Bibliography

A. Alin and S. Kurt. Testing non-additivity (interaction) in two-way ANOVA tables with no replication. *Statistical Methods in Medical Research*, 15:63–85, 2006. [p159, 165, 168]

F. Anscombe and J. Tukey. The examination and analysis of residuals. *Technometrics*, 5:141–160, 1963. [p159]

R. Biggs and R. Macmillan. The error of the red cell count. *Journal of Clinical Pathology*, 1:288–291, 1948. [p168, 170]

R. Boik. Testing additivity in two-way classifications with no replications: the locally best invariant test. *Journal of Applied Statistics*, 20:41–55, 1993a. [p159, 160, 168]

R. Boik. A comparison of three invariant tests of additivity in two-way classifications with no replications. *Computational Statistics and Data Analysis*, 15:411–424, 1993b. [p168]

O. Carter, B. Collier, and F. Davis. Blast furnace slags as agricultural liming materials. *Agronomy*, 43: 430–433, 1951. [p167, 170]

O. Davies and P. Goldsmith. *Statistical Methods in Research and Production*. Longman, 4th edition, 1972. [p168, 170]

C. Franck, D. Nielsen, and J. Osborne. A method for detecting hidden additivity in two-factor unreplicated experiments. *Computational Statistics & Data Analysis*, 67:95–104, 2013. [p159, 160, 162, 164, 168, 170]

F. Giesbrecht and M. Gumpertz. *Planning, Construction, and Statistical Analysis of Comparative Experiments*. Probability and Statistics. Wiley, 2004. [p168]

F. Graybill. Variance heterogeneity in a randomized block design. *Biometrics*, 10:516–520, 1954. [p164, 168, 170]

A. Hald. *Statistical Theory with Engineering Applications*. Statistics. Wiley Publications, 1952. [p168, 170]

J. Hartigan and M. Wong. A k-means clustering algorithm. *Applied Statistics*, 28:100–108, 1979. [p167]

C. Hirotsu. An approach to defining the pattern of interaction effects in a two-way layout. *Annals of the Institute of Statistical Mathematics*, 35:77–90, 1982. [p159]

D. Johnson and F. Graybill. An analysis of a two-way model with interaction and no replication. *Journal of the American Statistical Association*, 67:862–868, 1972. [p159, 160, 167]

M. Kharrati-Kopaei and S. Sadooghi-Alvandi. A new method for testing interaction in unreplicated two-way analysis of variance. *Communications in Statistics-Theory and Methods*, 36:2787–2803, 2007. [p159, 160, 166, 167, 168, 170]

M. Lentner and T. Bishop. *Experimental Design and Analysis*. Valley Book Company, second edition, 1993. [p168, 170]

J. MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1:281–297, 1967. [p167]

W. Malik, J. Mohring, and H. Piepho. A clustering-based test for non-additivity in an unreplicated two-way layout. *Communications in Statistics - Simulation and Computation*, 1:1–24, 2014. [p159, 160, 166, 167, 168]

J. Mandel. Non-additivity in two-way analysis of variance. *Journal of the American Statistical Association*, 56:878–888, 1961. [p159, 160, 165, 168, 170]

J. Mandel. A new analysis of variance model for non-additive data. *Technometrics*, 13:1–18, 1971. [p159, 168]

J. Mandel. *Evaluation and Control of Measurements*. Quality and Reliability Series. Marcel Dekker, 1991. [p168, 170]

N. Osborne, E. McKelvy, and H. Bearce. Density and thermal expansion of ethyl alcohol and its mixtures with water. *Bulletin of the Bureau of Standards*, 9, 1913. [p168, 170]

B. Ostle. *Statistics in Research, Basic Concepts and Techniques for Research Works*. The Iowa State University Press, 2nd edition, 1963. [p168, 170]

E. Ott and R. Snee. Identifying useful differences in a multiple-head machine. *JOURNAL OF QUALITY TECHNOLOGY*, 5:47–57, 1973. [p168, 170]

R. Ott and M. Longnecker. *An Introduction to Statistical Methods and Data Analysis*. Brooks/Cole, check edition, 2001. [p168, 170]

Y. Qiu. Antimicrobial susceptibility and clonal population structure of multidrug resistant *campylobacter jejuni* isolates from commercial turkeys in North Carolina. Master's thesis, North Carolina State University, 2013. [p159, 168, 170]

P. Simecek and M. Simeckova. Modification of tukey's additivity test. *Journal of Statistical Planning and Inference*, 143:1, 2012. [p160]

J. Tukey. One degree of freedom for non-additivity. *Biometrics*, 5:232–242, 1949. [p159, 160, 165, 168, 170]

F. Tusell. Testing for interaction in two-way ANOVA tables with no replication. *Computational Statistics and Data Analysis*, 10:29–45, 1990. [p159, 160]

Christopher Franck
Virginia Tech Department of Statistics
403 E Hutcheson Hall Blacksburg, VA 24061
United States of America
chfranck@vt.edu

Jason Osborne
North Carolina State University Department of Statistics
5238 SAS Hall Raleigh, NC 27695
United States of America
jason.osborne@ncsu.edu

Heteroscedastic Censored and Truncated Regression with crch

by Jakob W. Messner, Georg J. Mayr, and Achim Zeileis

Abstract The **crch** package provides functions for maximum likelihood estimation of censored or truncated regression models with conditional heteroscedasticity along with suitable standard methods to summarize the fitted models and compute predictions, residuals, etc. The supported distributions include left- or right-censored or truncated Gaussian, logistic, or student-t distributions with potentially different sets of regressors for modeling the conditional location and scale. The models and their R implementation are introduced and illustrated by numerical weather prediction tasks using precipitation data for Innsbruck (Austria).

Introduction

Censored or truncated response variables occur in a variety of applications. Censored data arise if exact values are only reported in a restricted range. Data may fall outside this range but are reported at the range limits. In contrast, if data outside this range are omitted completely we call the dataset truncated. E.g., consider wind measurements with an instrument that needs a certain minimum wind speed to start working. If wind speeds below this minimum are recorded as $\leq$ *minimum* the data are censored. If only wind speeds exceeding this limit are reported and those below are omitted the data are truncated. Even if the generating process is not as clear, censoring or truncation can be useful to consider limited data such as precipitation observations.

The tobit (Tobin, 1958) and truncated regression (Cragg, 1971) models are common linear regression models for censored and truncated conditionally normally distributed responses respectively. Beside truncated data, truncated regression is also used in two-part models (Cragg, 1971) for censored type data: a binary (e.g., probit) regression model fits the exceedance probability of the lower limit and a truncated regression model fits the value given the lower limit is exceeded.

Usually linear models like the tobit or truncated regression models assume homoscedasticity which means that the variance of an underlying normal distribution does not depend on covariates. However, sometimes this assumption does not hold and models that can consider conditional heteroscedasticity should be used. Such models have been proposed, e.g., for generalized linear models (Nelder and Pregibon, 1987; Smyth, 1989), generalized additive models (Rigby and Stasinopoulos, 1996, 2005), or beta regression (Cribari-Neto and Zeileis, 2010). There also exist several R packages with functions implementing the above models, e.g., **dglm** (Dunn and Smyth, 2014), **glmx** (Zeileis et al., 2013), **gamlss** (Rigby and Stasinopoulos, 2005), **betareg** (Grün et al., 2012) among others.

The **crch** package provides functions to fit censored and truncated regression models that consider conditional heteroscedasticity. It has a convenient interface to estimate these models with maximum likelihood and provides several methods for analysis and prediction. In addition to the typical conditional Gaussian distribution assumptions it also allows for logistic and student-t distributions with heavier tails.

In the following this paper presents the heteroscedastic censored and truncated regression models and their R implementation. Furthermore these models and their implementation are illustrated with numerical weather prediction data of precipitation in Innsbruck (Austria).

Regression models

For both, censored and truncated regression, a normalized latent response $(y^* - \mu)/\sigma$ is assumed to follow a certain distribution D

$$\frac{y^* - \mu}{\sigma} \sim D \tag{1}$$

The location parameter μ and a link function of the scale parameter $g(\sigma)$ are assumed to relate linearly to covariates $\mathbf{x} = (1, x_1, x_2, \ldots)^\top$ and $\mathbf{z} = (1, z_1, z_2, \ldots)^\top$:

$$\mu = \mathbf{x}^\top \beta \tag{2}$$

$$g(\sigma) = \mathbf{z}^\top \gamma \tag{3}$$

where $\beta = (\beta_0, \beta_1, \beta_2, \ldots)^\top$ and $\gamma = (\gamma_0, \gamma_1, \gamma_2, \ldots)^\top$ are coefficient vectors. The link function $g(\cdot) : \mathbb{R}^+ \mapsto \mathbb{R}$ is a strictly increasing and twice differentiable function; e.g., the logarithm (i.e.,

$g(\sigma) = \log(\sigma)$) is a well suited function. Although they only map to $\mathbb{R}^+$, the identity $g(\sigma) = \sigma$ or the quadratic function $g(\sigma) = \sigma^2$ can be useful as well. However, problems in the numerical optimization can occur.

Commonly D is the standard normal distribution so that y^* is assumed to be normally distributed with mean μ and variance σ^2. D might also be assumed to be a standard logistic or a student-t distribution if heavier tails are required. The tail weight of the student-t distribution can be controlled by the degrees of freedom ν which can either be set to a certain value or estimated as an additional parameter. To assure positive values, $\log(\nu)$ is modeled in the latter case.

$$\log(\nu) = \delta \tag{4}$$

Censored regression (tobit)

The exact values of censored responses are only known in an interval defined by *left* and *right*. Observation outside this interval are mapped to the interval limits

$$y = \begin{cases} left & y^* \leq left \\ y^* & left < y^* < right \\ right & y^* \geq right \end{cases} \tag{5}$$

The coefficients β, γ, and δ (Equations 2–4) can be estimated by maximizing the sum over the data set of the log-likelihood function $\log(f_{cens}(y, \mu, \sigma))$, where

$$f_{cens}(y, \mu, \sigma) = \begin{cases} F\left(\frac{left-\mu}{\sigma}\right) & y \leq left \\ f\left(\frac{y-\mu}{\sigma}\right) & left < y < right \\ \left(1 - F\left(\frac{right-\mu}{\sigma}\right)\right) & y \geq right \end{cases} \tag{6}$$

$F()$ and $f()$ are the cumulative distribution function and the probability density function of D, respectively. If D is the normal distribution this model is a heteroscedastic variant of the tobit model (Tobin, 1958).

Truncated regression

Truncated responses occur when latent responses below or above some thresholds are omitted.

$$y = y^* | left < y^* < right \tag{7}$$

Then y follows a truncated distribution with probability density function

$$f_{tr}(y, \mu, \sigma) = \frac{f\left(\frac{y-\mu}{\sigma}\right)}{F\left(\frac{right-\mu}{\sigma}\right) - F\left(\frac{left-\mu}{\sigma}\right)} \tag{8}$$

In that case the coefficients β, γ, and δ can be estimated by maximizing the sum over the data set of the log-likelihood function

$$\log(f_{tr}(y, \mu, \sigma)) \tag{9}$$

R implementation

The models from the previous section can both be fitted with the `crch()` function provided by the **crch** package. This function takes a formula and data, sets up the likelihood function, gradients and Hessian matrix and uses `optim()` to maximize the likelihood. It returns an S3 object for which various standard methods are available. We tried to build an interface as similar to `glm()` as possible to facilitate the usage.

```
crch(formula, data, subset, na.action, weights, offset, link.scale = "log",
  dist = "gaussian", df = NULL, left = -Inf, right = Inf, truncated = FALSE,
  control = crch.control(...), model = TRUE, x = FALSE, y = FALSE, ...)
```

Here `formula`, `data`, `na.action`, `weights`, and `offset` have their standard model frame meanings (e.g., Chambers and Hastie, 1992). However, as provided in the **Formula** package (Zeileis and Croissant, 2010) `formula` can have two parts separated by '`|`' where the first part defines the location model

Function	Description
`print()`	Print function call and estimated coefficients.
`summary()`	Standard regression output (coefficient estimates, standard errors, partial Wald tests). Returns an object of class `"summary.crch"` containing summary statistics which has a `print()` method.
`coef()`	Extract model coefficients where `model` specifies whether a single vector containing all coefficients (`"full"`) or the coefficients for the location (`"location"`), scale (`"scale"`) or degrees of freedom (`"df"`) are returned.
`vcov()`	Variance-covariance matrix of the estimated coefficients.
`predict()`	Predictions for new data where `"type"` controls whether location (`"response"`/`"location"`), scale (`"scale"`) or quantiles (`"quantile"`) are predicted. Quantile probabilities are specified by `at`.
`fitted()`	Fitted values for observed data where `"type"` controls whether location (`"location"`) or scale (`"scale"`) values are returned.
`residuals()`	Extract various types of residuals where `type` can be `"standardized"` (default), `"pearson"`, `"response"`, or `"quantile"`.
`terms()`	Extract terms of model components.
`logLik()`	Extract fitted log-likelihood.

Table 1: Functions and methods for objects of class `"crch"`.

and the second part the scale model. E.g., with `y ~ x1 + x2 | z1 + z2` the location model is specified by `y ~ x1 + x2` and the scale model by `~ z1 + z2`. Known offsets can be specified for the location model by `offset` or for both, the location and scale model, inside `formula`, i.e., `y ~ x1 + x2 + offset(x3) | z1 + z2 + offset(z3)`.

The link function $g(\cdot)$ for the scale model can be specified by `link.scale`. The default is `"log"`, also supported are `"identity"` and `"quadratic"`. Furthermore, an arbitrary link function can be specified by supplying an object of class `"link-glm"` containing `linkfun`, `linkinv`, `mu.eta`, and `name`. Furthermore it must contain the second derivative `dmu.deta` if analytical Hessians are employed.

`dist` specifies the used distribution. Currently supported are `"gaussian"` (the default), `"logistic"`, and `"student"`. If `dist = "student"` the degrees of freedom can be set by the `df` argument. If set to `NULL` (the default) the degrees of freedom are estimated by maximum likelihood (Equation 4).

`left` and `right` define the lower and upper censoring or truncation points respectively. The logical argument `truncated` defines whether a censored or truncated model is estimated. Note that also a wrapper function `trch()` exists that is equivalent to `crch()` but with default `truncated = TRUE`.

The maximum likelihood estimation is carried out with the R function `optim()` using control options specified in `crch.control()`. By default the `"BFGS"` method is applied. If no starting values are supplied, coefficients from `lm()` are used as starting values for the location part. For the scale model the intercept is initialized with the link function of the residual standard deviation from `lm()` and the remaining scale coefficients are initialized with 0. If the degrees of freedom of a student-t distribution are estimated they are initialized by 10. For the student-t distribution with estimated degrees of freedom the covariance matrix estimate is derived from the numerical Hessian returned by `optim()`. For fixed degrees of freedom and Gaussian and logistic distributions the covariance matrix is derived analytically. However, by setting `hessian = TRUE` the numerical Hessian can be employed for those models as well.

Finally `model`, `y`, and `x` specify whether the model frame, response, or model matrix are returned.

The returned model fit of class `"crch"` is a list similar to `"glm"` objects. Some components like `coefficients` are lists with elements for location, scale, and degrees of freedom. The package also provides a set of extractor methods for `"crch"` objects that are listed in Table 1.

Additional to the `crch()` function and corresponding methods the **crch** package also provides probability density, cumulative distribution, random number, and quantile functions for censored and truncated normal, logistic, and student-t distributions. Furthermore it also provides a function `hxlr()` (heteroscedastic extended logistic regression) to fit heteroscedastic interval-censored regression models (Messner et al., 2014c).

Note that alternatives to `crch()` heteroscedastic censored and truncated models could also be

fitted by the R package **gamlss** (Rigby and Stasinopoulos, 2005) with the add-on packages **gamlss.cens** and **gamlss.tr**. However, for the special case of linear censored truncated regression models with the Gaussian, logistic, or student-t distribution **crch** provides a fast and convenient interface and various useful methods for analysis and prediction.

Example

This section shows a weather forecast example application of censored and truncated regression models fitted with `crch()`. Weather forecasts are usually based on numerical weather prediction (NWP) models that take the current state of the atmosphere and compute future weather by numerically simulating the most important atmospheric processes. However, because of uncertain initial conditions and unknown or unresolved processes these numerical predictions are always subject to errors. To estimate these errors, many weather centers provide so called ensemble forecasts: several NWP runs that use different initial conditions and model formulations. Unfortunately these ensemble forecasts cannot consider all error sources so they are often still biased and uncalibrated. Thus they are often calibrated and corrected for systematic errors by statistical post-processing.

One popular post-processing method is heteroscedastic linear regression where the ensemble mean is used as regressor for the location and the ensemble standard deviation or variance is used as regressor for the scale (e.g., Gneiting et al., 2005). Because not all meteorological variables can be assumed to be normally distributed this idea has also been extended to other distributions including truncated regression for wind (Thorarinsdottir and Gneiting, 2010) and censored regression for wind power (Messner et al., 2014b) or precipitation (Messner et al., 2014a).

The following example applies heteroscedastic censored regression with a logistic distribution assumption to precipitation data in Innsbruck (Austria). Furthermore, a two-part model tests whether the occurrence of precipitation and the precipitation amount are driven by the same process.

First, the **crch** package is loaded together with an included precipitation data set with forecasts and observations for Innsbruck (Austria)

```
R> library("crch")
R> data("RainIbk", package = "crch")
```

The `data.frame` `RainIbk` contains observed 3 day-accumulated precipitation amounts (`rain`) and the corresponding 11 member ensemble forecasts of total accumulated precipitation amount between 5 and 8 days in advance (`rainfc.1`, `rainfc.2`, ... `rainfc.11`). The rownames are the end date of the 3 days over which the precipitation amounts are accumulated respectively; i.e., the respective forecasts are issued 8 days before these dates.

In previous studies it has been shown that it is of advantage to model the square root of precipitation rather than precipitation itself. Thus all precipitation amounts are square rooted before ensemble mean and standard deviation are derived. Furthermore, events with no variation in the ensemble are omitted:

```
R> RainIbk <- sqrt(RainIbk)
R> RainIbk$ensmean <- apply(RainIbk[,grep('^rainfc',names(RainIbk))], 1, mean)
R> RainIbk$enssd <- apply(RainIbk[,grep('^rainfc',names(RainIbk))], 1, sd)
R> RainIbk <- subset(RainIbk, enssd > 0)
```

A scatterplot of `rain` against `ensmean`

```
R> plot(rain ~ ensmean, data = RainIbk, pch = 19, col = gray(0, alpha = 0.2))
R> abline(0,1, col = "red")
```

indicates a linear relationship that differs from a 1-to-1 relationship (Figure 1). Precipitation is clearly non-negative with many zero observations. Thus censored regression or a two-part model are suitable to estimate this relationship.

First we fit a logistic censored model for `rain` with `ensmean` as regressor for the location and `log(enssd)` as regressor for the scale.

```
R> CRCH <- crch(rain ~ ensmean | log(enssd), data = RainIbk, left = 0,
+    dist = "logistic")
R> summary(CRCH)

Call:
crch(formula = rain ~ ensmean | log(enssd), data = RainIbk,
    dist = "logistic", left = 0)
```

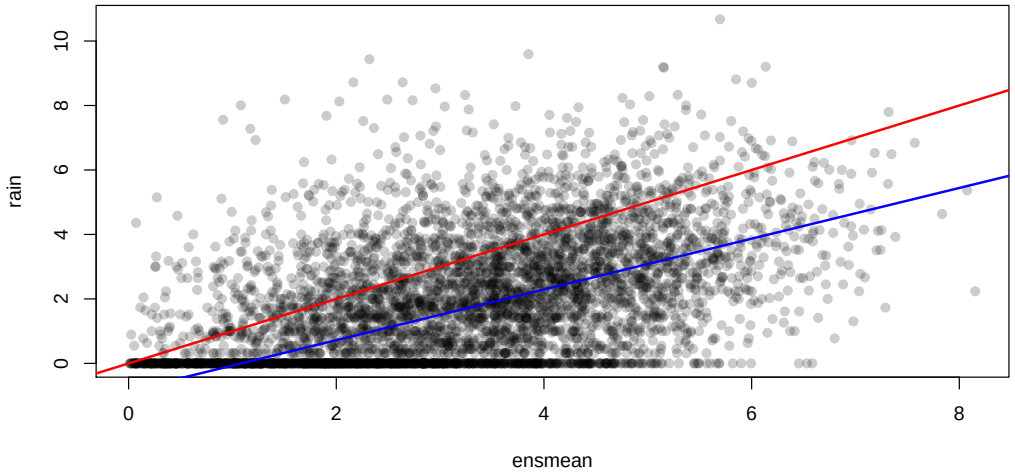


Figure 1: Square rooted precipitation amount against ensemble mean forecasts. A line with intercept 0 and slope 1 is shown in red and the censored regression fit in blue.

```
Standardized residuals:
    Min      1Q  Median      3Q     Max
-3.5780 -0.6554  0.1673  1.1189  7.4990

Coefficients (location model):
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.85266    0.06903  -12.35   <2e-16 ***
ensmean      0.78686    0.01921   40.97   <2e-16 ***

Coefficients (scale model with log link):
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.11744    0.01460   8.046 8.58e-16 ***
log(enssd)   0.27055    0.03503   7.723 1.14e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Distribution: logistic
Log-likelihood: -8921 on 4 Df
Number of iterations in BFGS optimization: 15
```

Both, `ensmean` and `log(enssd)` are highly significant according to the Wald test performed by the `summary()` method. The location model is also shown in Figure 1:

```
R> abline(coef(CRCH)[1:2], col = "blue")
```

If we compare this model to a constant scale model (tobit model with logistic distribution)

```
R> CR <- crch(rain ~ ensmean, data = RainIbk, left = 0, dist = "logistic")
R> cbind(AIC(CR, CRCH), BIC = BIC(CR, CRCH)[,2])

     df      AIC      BIC
CR    3 17905.69 17925.22
CRCH  4 17850.30 17876.33
```

we see that the scale model clearly improves the fit regarding AIC and BIC.

A comparison of the logistic model with a Gaussian and a student-t model

```
R> CRCHgau <- crch(rain ~ ensmean | log(enssd), data = RainIbk, left = 0,
+    dist = "gaussian")
R> CRCHstud <- crch(rain ~ ensmean | log(enssd), data = RainIbk, left = 0,
+    dist = "student")
R> AIC(CRCH, CRCHgau, CRCHstud)
```

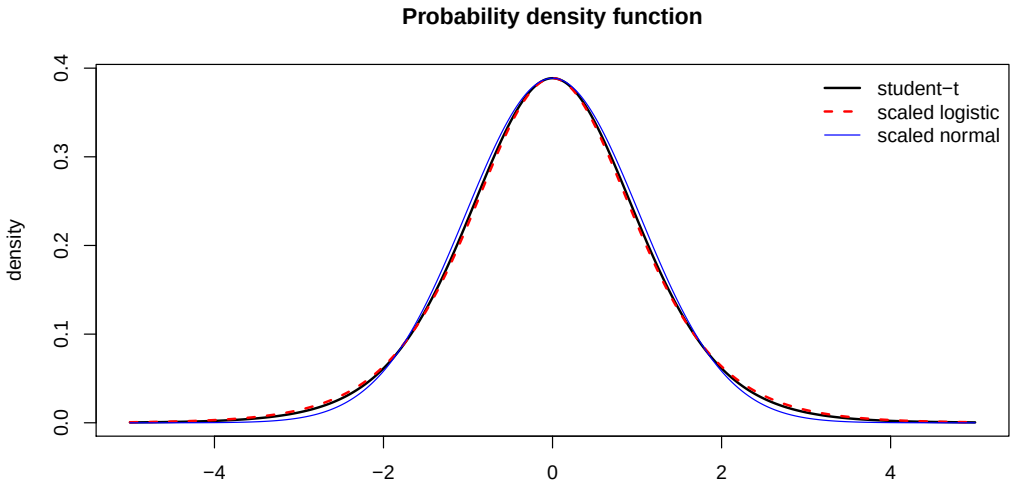


Figure 2: Probability density functions of a student-t distribution with 9.56 degrees of freedom, a logistic, and a normal distribution. The densities of the logistic and normal distribution are scaled to facilitate comparison.

```
         df      AIC
CRCH      4 17850.30
CRCHgau   4 17897.23
CRCHstud  5 17850.65
```

confirms the logistic distribution assumption. Note, that with the estimated degrees of freedom of 9.56 the student-t distribution resembles the (scaled) logistic distribution quite well (see Figure 2).

In the censored model the occurrence of precipitation and precipitation amount are assumed to be driven by the same process. To test this assumption we compare the censored model with a two-part model consisting of a heteroscedastic logit model and a truncated regression model with logistic distribution assumption. For the heteroscedastic logit model we use `hetglm()` from the **glmx** package and for the truncated model we employ the `crch()` function with the argument `truncated = TRUE`.

```
R> library("glmx")
R> BIN <- hetglm(I(rain > 0) ~ ensmean | log(enssd), data = RainIbk,
+           family = binomial(link = "logit"))
R> TRCH <- crch(rain~ensmean | log(enssd), data = RainIbk, subset = rain > 0,
+           left = 0, dist = "logistic", truncated = TRUE)
```

In the heteroscedastic logit model, the intercept of the scale model is not identified. Thus, the location coefficients of the censored and truncated regression models have to be scaled to compare them with the logit model.

```
R> cbind("CRCH" = c(coef(CRCH, "location")/exp(coef(CRCH, "scale"))[1],
+          coef(CRCH, "scale")[2]),
+        "BIN" = coef(BIN),
+        "TRCH" = c(coef(TRCH, "location")/exp(coef(TRCH, "scale"))[1],
+          coef(TRCH, "scale")[2]))

                  CRCH        BIN      TRCH
(Intercept) -0.7581811 -1.0181715 0.2635421
ensmean      0.6996699  0.7789091 0.5455966
log(enssd)   0.2705476  0.4539908 0.2326229
```

The different (scaled) coefficients indicate that different processes drive the occurrence of precipitation and precipitation amount. This is also confirmed by AIC and BIC that are clearly better for the two-part model than for the censored model:

```
R> loglik <- c("Censored" = logLik(CRCH), "Two-Part" = logLik(BIN) + logLik(TRCH))
R> df <- c(4, 7)
```

```
R> aic <- -2 * loglik + 2 * df
R> bic <- -2 * loglik + log(nrow(RainIbk)) * df
R> cbind(df, AIC = aic, BIC = bic)

          df      AIC      BIC
Censored  4 17850.30 17876.33
Two-Part  7 17744.82 17790.39
```

Finally, we can use the fitted models to predict future precipitation. Therefore assume that the current NWP forecast of square rooted precipitation has an ensemble mean of 1.8 and an ensemble standard deviation of 0.9. A median precipitation forecast of the censored model can then easily be computed with

```
R> newdata <- data.frame(ensmean = 1.8, enssd = 0.9)
R> predict(CRCH, newdata, type = "quantile", at = 0.5)^2

        1
0.3177399
```

Note, that the prediction has to be squared since all models fit the square root of precipitation. In the two-part model the probability to stay below a threshold q is composed of

$$P(y \leq q) = 1 - P(y > 0) + P(y > 0) \cdot P(y \leq q | y > 0) \tag{10}$$

Thus median precipitation equals the $(P(y > 0) - 0.5)/P(y > 0)$-quantile of the truncated distribution.

```
R> p <- predict(BIN, newdata)
R> predict(TRCH, newdata, type = "quantile", at = (p - 0.5)/p)^2

      1.1
0.4156972
```

Probabilities to exceed, e.g., 5mm can be predicted with cumulative distribution functions (e.g., `pclogis()`, `ptlogis()`) that are also provided in the **crch** package.

```
R> mu <- predict(CRCH, newdata, type = "location")
R> sigma <- predict(CRCH, newdata, type = "scale")
R> pclogis(sqrt(5), mu, sigma, lower.tail = FALSE, left = 0)

[1] 0.177983

R> mu <- predict(TRCH, newdata, type = "location")
R> sigma <- predict(TRCH, newdata, type = "scale")
R> p * ptlogis(sqrt(5), mu, sigma, lower.tail = FALSE, left = 0)

        1
0.2108671
```

Note, that `pclogis()` could also be replaced by `plogis()` since they are equivalent between *left* and *right*.

Clearly, other types of model misspecification or model generalization (depending on the point of view) for the classical tobit model are possible. In addition to heteroscedasticity, the type of response distribution, and the presence of hurdle effects as explored in the application here, further aspects might have to be addressed by the model. Especially in economics and the social sciences sample selection effects might be present in the two-part model which can be addressed (in the homoscedastic normal case) using the R packages **sampleSelection** (Toomet and Henningsen, 2008) or **mhurdle** (Croissant et al., 2013). Furthermore, the scale link function or potential nonlinearities in the regression functions could be assessed, e.g., using the **gamlss** suite of packages (Stasinopoulos and Rigby, 2007).

Summary

Censored and truncated response models are common in econometrics and other statistical applications. However, often the homoscedasticity assumption of these models is not fulfilled. This paper presented the **crch** package that provides functions to fit censored or truncated regression models with conditional heteroscedasticity. It supports Gaussian, logistic or student-t distributed censored or truncated responses and provides various convenient methods for analysis and prediction. To illustrate the package we showed that heteroscedastic censored and truncated regression models are well suited to improve precipitation forecasts.

Bibliography

J. M. Chambers and T. J. Hastie. *Statistical Models in S*. Chapman & Hall, London, 1992. [p174]

J. G. Cragg. Some statistical models for limited dependent variables with application to the demand for durable goods. *Econometrica*, 39(5):829–844, 1971. doi: 10.2307/1909582. [p173]

F. Cribari-Neto and A. Zeileis. Beta regression in R. *Journal of Statistical Software*, 34(2):1–24, 2010. doi: 10.18637/jss.v034.i02. [p173]

Y. Croissant, F. Carlevaro, and S. Hoareau. *mhurdle: Multiple Hurdle Tobit Models*, 2013. URL http://CRAN.R-project.org/package=mhurdle. R package version 1.0-1. [p179]

P. K. Dunn and G. K. Smyth. *dglm: Double Generalized Linear Models*, 2014. URL http://CRAN.R-project.org/package=dglm. R package version 1.8.1. [p173]

T. Gneiting, A. E. Raftery, A. H. Westveld, and T. Goldman. Calibrated probabilistic forecasting using ensemble model output statistics and minimum CRPS estimation. *Monthly Weather Review*, 133(5): 1098–1118, 2005. doi: 10.1175/MWR2904.1. [p176]

B. Grün, I. Kosmidis, and A. Zeileis. Extended beta regression in R: Shaken, stirred, mixed, and partitioned. *Journal of Statistical Software*, 48(11):1–25, 2012. doi: 10.18637/jss.v048.i11. [p173]

J. W. Messner, G. J. Mayr, D. S. Wilks, and A. Zeileis. Extending extended logistic regression: Extended vs. separate vs. ordered vs. censored. *Monthly Weather Review*, 142:3003–3014, 2014a. doi: 10.1175/MWR-D-13-00355.1. [p176]

J. W. Messner, A. Zeileis, J. Broecker, and G. J. Mayr. Probabilistic wind power forecasts with an inverse power curve transformation and censored regression. *Wind Energy*, 17(11):1753–1766, 2014b. doi: 10.1002/we.1666. [p176]

J. W. Messner, A. Zeileis, G. J. Mayr, and D. S. Wilks. Heteroscedastic extended logistic regression for post-processing of ensemble guidance. *Monthly Weather Review*, 142:448–456, 2014c. doi: 10.1175/MWR-D-13-00271.1. [p175]

J. A. Nelder and D. Pregibon. An extended quasi-likelihood function. *Biometrika*, 74(2):221–232, 1987. doi: 10.2307/2336136. [p173]

R. A. Rigby and D. M. Stasinopoulos. Mean and dispersion additive models. In W. Härdle and M. G. Schimek, editors, *Statistical Theory and Computational Aspects of Smoothing*, Contributions to Statistics, pages 215–230. Physica-Verlag, 1996. doi: 10.1007/978-3-642-48425-4_16. [p173]

R. A. Rigby and D. M. Stasinopoulos. Generalized additive models for location, scale and shape. *Journal of the Royal Statistical Society C*, 54(3):507–554, 2005. doi: 10.1111/j.1467-9876.2005.00510.x. [p173, 176]

G. K. Smyth. Generalized linear models with varying dispersion. *Journal of the Royal Statistical Society B*, 51(1):47–60, 1989. [p173]

D. Stasinopoulos and R. Rigby. Generalized additive models for location scale and shape (GAMLSS) in R. *Journal of Statistical Software*, 23(7):1–46, 2007. doi: 10.18637/jss.v023.i07. [p179]

T. L. Thorarinsdottir and T. Gneiting. Probabilistic forecasts of wind speed: Ensemble model output statistics by using heteroscedastic censored regression. *Journal of the Royal Statistical Society A*, 173 (2):371–388, 2010. doi: 10.1111/j.1467-985X.2009.00616.x. [p176]

J. Tobin. Estimation of relationships for limited dependent variables. *Econometrica*, 26(1):24–36, 1958. doi: 10.2307/1907382. [p173, 174]

O. Toomet and A. Henningsen. Sample selection models in R: Package sampleSelection. *Journal of Statistical Software*, 27(7):1–23, 2008. doi: 10.18637/jss.v027.i07. [p179]

A. Zeileis and Y. Croissant. Extended model formulas in R: Multiple parts and multiple responses. *Journal of Statistical Software*, 34(1):1–13, 2010. doi: 10.18637/jss.v034.i01. [p174]

A. Zeileis, R. Koenker, and P. Doebler. *glmx: Generalized Linear Models Extended*, 2013. URL http://CRAN.R-project.org/package=glmx. R package version 0.1-0. [p173]

Jakob W. Messner
Universität Innsbruck
6020 Innsbruck, Austria
jakob.messner@uibk.ac.at

Georg J. Mayr
Universität Innsbruck
6020 Innsbruck, Austria
georg.mayr@uibk.ac.at

Achim Zeileis
Universität Innsbruck
6020 Innsbruck, Austria
achim.zeileis@uibk.ac.at

Model Builder for Item Factor Analysis with OpenMx

by Joshua N. Pritikin and Karen M. Schmidt

Abstract We introduce a shiny web application to facilitate the construction of Item Factor Analysis (a.k.a. Item Response Theory) models using the **OpenMx** package. The web application assists with importing data, outcome recoding, and model specification. However, the app does not conduct any analysis but, rather, generates an analysis script. Generated Rmarkdown output serves dual purposes: to analyze a data set and demonstrate good programming practices. The app can be used as a teaching tool or as a starting point for custom analysis scripts.

An overview of OpenMx

OpenMx, a modular package originally designed for structural equation modeling (Neale et al., in press), recently gained the ability to handle Item Factor Analysis (a.k.a. Item Response Theory, Modern Test Theory) models (Pritikin et al., 2015). Although a goal of **OpenMx** is to cater to the statistical power user and facilitate analyses that are difficult to conduct in other software, the development team is always on the lookout for ways to ease the learning curve for novice users as well. Here we introduce a new **shiny** (RStudio and Inc., 2014) web application to generate **OpenMx** code in **Rmarkdown** format (Allaire et al., 2014). We believe this code generator substantially lowers the barrier to entry for novice users of Item Factor Analysis (IFA) and encourages a culture of literate programming (Knuth, 1984) and reproducible science (Peng, 2011; Nosek et al., 2015). The generated code can be customized at many levels. This flexibility enables the production of custom analyses and reports as users grow more sophisticated in their modeling expectations.

The statistical model

Item analysis is concerned with items that are scored correct/incorrect or on an ordinal scale. Many psychological surveys use an ordinal scale. For example, participants may be asked to respond to an item like, "I feel I am in charge of the situation in which I live." on a 5-point Likert scale from *agree* to *disagree*. Whether dichotomous or ordinal, the conditional likelihood of response x_{ij} to item j from person i with item parameters ξ_j and latent ability (a.k.a. latent trait) θ_i is

$$L(x_i|\xi,\theta_i) = \prod_j \Pr(\text{pick} = x_{ij}|\xi_j,\theta_i). \tag{1}$$

One implication of Equation 1 is that items are assumed to be conditionally independent given the latent ability θ_i. That is, the outcome of one item does not have any influence on another item after controlling for ξ and θ_i. The unconditional likelihood is obtained by integrating over the latent distribution θ_i,

$$L(x_i|\xi) = \int L(x_i|\xi,\theta_i)L(\theta_i)\mathrm{d}\theta_i. \tag{2}$$

With an assumption that examinees are independently and identically distributed, we can sum the individual log likelihoods,

$$\mathcal{L} = \sum_i \log L(x_i|\xi). \tag{3}$$

Optimization consists of finding the ξ that maximizes this function. **OpenMx** presently offers only one choice for optimization, an Expectation-Maximization algorithm using equal interval quadrature to evaluate the integral in Equation 2 (Bock and Aitkin, 1981). In the future, we plan to add complementary algorithms such as Metropolis-Hastings Robbins-Monro, that is more efficient at optimizing certain problems (Cai, 2010b).

Several models are readily available to plug in as the response probability function $\Pr(\text{pick} = x_{ij}|\xi_j,\theta_i)$ in Equation 1. All of these response probability functions are built from the logistic function,

$$\text{logistic}(l) \equiv \text{logit}^{-1}(l) \equiv \frac{1}{1+\exp(-l)}.$$

Details of the parameterizations are given here. A discussion of these item models more appealing to intuition is given in the next section.

Dichotomous model

The dichotomous response probability function can model items when there are exactly two possible outcomes. It is defined as,

$$\Pr(\text{pick} = 0|a, b, g, u, \tau) = 1 - \Pr(\text{pick} = 1|a, b, g, u, \tau) \tag{4}$$

$$\Pr(\text{pick} = 1|a, b, g, u, \tau) = \text{logit}^{-1}(g) + (\text{logit}^{-1}(u) - \text{logit}^{-1}(g))\frac{1}{1 + \exp(-(a\tau + b))} \tag{5}$$

where a is the slope, b is the intercept, g is the pseudo-guessing lower asymptote expressed in logit units, u is the upper asymptote expressed in logit units, and τ is the latent ability of the examinee (Birnbaum, 1968; Loken and Rulison, 2010). A #*PL* naming shorthand has been developed to refer to versions of the dichotomous model with different numbers of free parameters. Model nPL refers to the model obtained by freeing the first n of parameters b, a, g, and u.

Graded response model

The graded response model is a response probability function for 2 or more outcomes (Samejima, 1969; Cai, 2010b). For outcomes k in 0 to K, slope vector a, intercept vector b, and latent ability vector τ, it is defined as,

$$\Pr(\text{pick} = K|a, b, \tau) = \frac{1}{1 + \exp(-(a\tau + b_K))} \tag{6}$$

$$\Pr(\text{pick} = k|a, b, \tau) = \frac{1}{1 + \exp(-(a\tau + b_k))} - \frac{1}{1 + \exp(-(a\tau + b_{k+1}))} \tag{7}$$

$$\Pr(\text{pick} = 0|a, b, \tau) = 1 - \Pr(\text{pick} = 1|a, b_1, \tau). \tag{8}$$

Nominal model

The nominal model is a response probability function for items with 3 or more outcomes (e.g., Thissen et al., 2010). It can be defined as,

$$a = T_a\alpha \tag{9}$$

$$c = T_c\gamma \tag{10}$$

$$\Pr(\text{pick} = k|s, a_k, c_k, \tau) = C\,\frac{1}{1 + \exp(-(s\tau a_k + c_k))} \tag{11}$$

where a_k and c_k are the result of multiplying two vectors of free parameters α and γ by fixed matrices T_a and T_c, respectively; a_0 and c_0 are fixed to 0 for identification; s is the per-item slope; and C is a normalizing constant to ensure that $\sum_k \Pr(\text{pick} = k) = 1$.

Item models

Modern test theory employs item models, $\Pr(\text{pick} = x_{ij}|\zeta_j, \theta_i)$ (from Equation 1). To better appreciate how modern test theory works, it is helpful to develop an intuitive understanding of item models. The essential idea is the conversion of ordinal (or dichotomous) data into continuous data conditional on examinee skill. In Figure 1, the black dots represent the dichotomous data. Here we assume that examinee skill is known so that we can plot the black dots at the appropriate place on the x axis. The next step is to partition the x axis into equal interval bins. The proportion of examinees who responded correctly is displayed in blue in the middle of each bin. These blue numbers are our new continuous data, conditional on examinee skill. While we assumed that examinee skill was known, this assumption is actually unnecessary. The optimization algorithm can

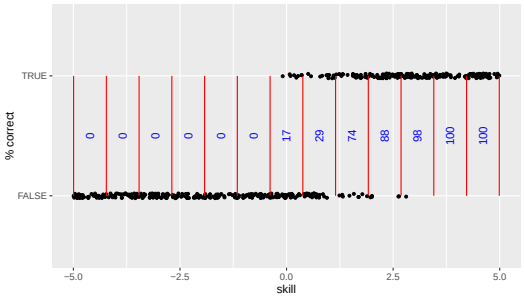


Figure 1: Dichotomous data converted into continuous data conditional on examinee skill.

make a rough estimate of examinee skill, proceed to improve the model, and repeat this process until change is less than some epsilon.

To further inform your intuition about item models, it can be helpful to place yourself in the position of the optimization algorithm. Enter the following commands to launch the model explorer tool and browse to the output web server address. It is possible to do this without **RStudio**, but **RStudio** makes everything easier so we recommend using **RStudio**. Note that the port number (`3726` printed below) may be different on your computer.

```
> library(ifaTools)
> itemModelExplorer()

Listening on http://127.0.0.1:3726
```

Your browser should show a screen similar to Figure 2. Try experimenting with all the controls. Early in the development of item models, model parameters closely corresponded to the psychological concepts of *difficulty* and *discrimination* (Birnbaum, 1968). For example, difficult items are only answered correctly by the brightest examinees while most examinees may correctly answer easy items. *Discrimination* quantifies how much we learn from a given response. Well-designed items discriminate examinee skill. The causes of poor item discrimination are many. An item may be hurt by awkward wording, by asking examinees something that is somewhat off-topic, or by asking the same question in slightly different ways.

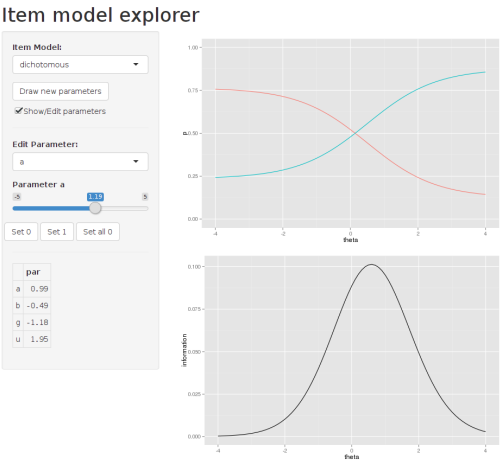


Figure 2: Item model explorer with the dichotomous model selected. The upper plot exhibits the model predicted chance of outcomes conditional on the latent trait (theta). The lower plot exhibits the theoretical item information conditional on the latent trait.

Some item model parameters still retain a close connection to difficulty and discrimination. For example, the dichotomous model's a parameter corresponds with discrimination and the negative b parameter divided by a corresponds with difficulty (Equation 5). However, as item models have grown more flexible, the parameter values and intuitive interpretation have become more distant. To understand item models in general, it is helpful to plot category curves and information by the latent trait (Figure 2). Some examples of latent traits which can be measured in this framework are mathematical skill, vocabulary, or sleep quality.

The way to interpret these plots is to start by picking a value for the latent trait. Suppose we know that examinee Alice has a latent ability of 2 logit units. If we trace across the plot where the x axis is 2 then we find that Alice has a 75% chance of getting the item correct (blue curve) and a 25% chance of getting it incorrect (red curve). In addition, we find that this item will give us 0.05 units of information about Alice (black curve). The difficulty of the item is where the correct and incorrect curves cross at about 0.2 logits. The discrimination of the item is given by the information plot. This item provides more information about examinees with latent skill between -1 and 2 than elsewhere on the skill continuum.

Much can be gleaned about item models by inspection of these plots. However, it is worth conveying a few additional details specific to particular item models. The dichotomous model's g and u asymptote parameters are in logit units. To transform these values back into probabilities use R's `plogis` function. The g parameter may represent the chance of an examinee guessing the item correctly. This parameter is also often called the pseudo-guessing parameter due to the probability of a low ability examinee getting an item correct at a non-zero asymptote. The u parameter, or upper asymptote parameter, may represent the chance of an examinee committing a careless mistake, reflecting high ability examinee behavior. In this case, the upper asymptote never reaches one (Loken and Rulison, 2010).

By default, the nominal model uses *trend* for the `T.a` and `T.c` matrices (Equation 10). This parameterization is also known as the Fourier basis. The effect is that the `alf` and `gam` parameters control the lowest frequency variation to the highest frequency variation. To develop an intuition for how this works, set all parameters to zero then set `a`, `alf1` and `gam2` to 1. Experiment with the `gam` parameters before you experiment with the `alf` parameters. Refer to Thissen et al. (2010) for discussion of the

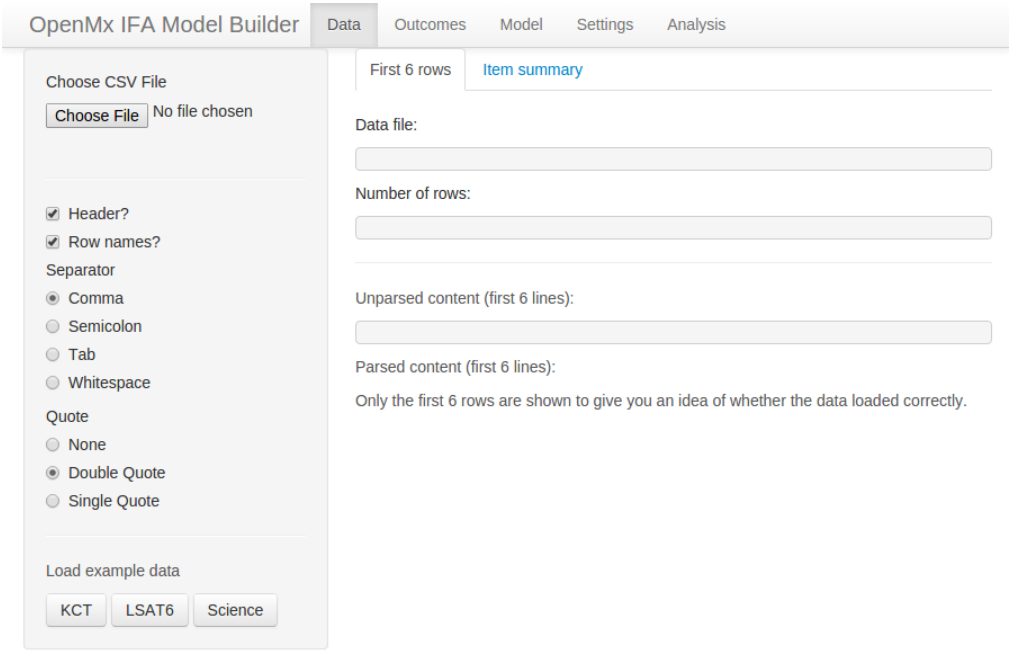


Figure 3: Initial screen shown after start up.

possibilities of this item model. Custom `T.a` and `T.c` matrices are not available in the model explorer app, but can be specified in R code.

The "`Show/Edit parameters`" checkbox has a special didactic purpose. Open two copies of the item model explorer. On one copy, un-check the "`Show/Edit parameters`" checkbox to hide the parameters and click the "`Draw new parameters`" button. On the second copy of the item model explorer, adjust the item model parameters to try to match the plot produced on the first item model explorer. You can check your answers by checking the "`Show/Edit parameters`" checkbox. When you play this game, you are doing part of what the optimization algorithm does when it fits models to data. Note that there is no need to practice this skill. The computer will do it for you.

The model builder

Enter the following commands to launch the model builder tool and browse to the output web server address. As before, it is possible to do this without **RStudio**, but **RStudio** makes everything easier so we recommend using **RStudio**. Note that the port number (`3726` printed below) may be different on your computer.

```
> library(ifaTools)
> modelBuilder()

Listening on http://127.0.0.1:3726
```

Your browser should show a screen similar to Figure 3. Take care not to hit the `Reload` button because that will reset the app. Learn how to save your work (detailed below) before you experiment with the `Reload` button. Across the top are tabs that organize the major functions of the model builder app. On the left side is a control panel for the currently selected tab `Data`. Example data sets are available at the bottom of the control panel. You are welcome to experiment with these, but we will focus on the process of loading an external data set. If you prefer to follow along with a video then browse to `http://youtu.be/xHeb5_CWnCk` for dichotomous data and `http://youtu.be/iwtpleltteQ` for polytomous data.

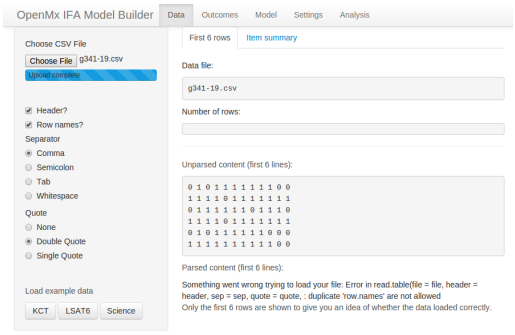


Figure 4: After loading the `g341-19.csv` data.

Dichotomous data

Click on the "Choose File" button[1] and select g341-19.csv, a dichotomous data set that is available in the **ifaTools** package (Pritikin, 2015a). The first 6 lines will appear in the "Unparsed content" section (see Figure 4).[2] This makes it easy to see how the file is formatted. The "Parsed content" section reports an error. By reading the error carefully, you will find mention that "duplicate 'row.names' are not allowed." Since "Row names?" is enabled in the control panel, the model builder app expects the first column of data to be dedicated to row names. A row name is typically the examinee's name or numeric identifier. A glance at the unparsed content reveals that no row names have been given in this data set.

Click the "Row names?" checkbox in the control panel to disable row names. Immediately (without reloading the data), the error message in the "Parsed content" section will be replaced by some of the data organized into a single column. The column name will read X010111111100. A column name like that should raise suspicion. Since the "Header?" checkbox is enabled in the control panel, the model builder app expects the first line of the data to contain column names. Therefore, the first line of data is misinterpreted.

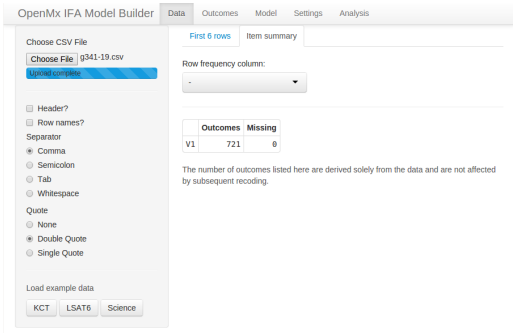


Figure 5: A summary of the g341-19.csv data set when parsed incorrectly as a single column.

Click the "Header?" checkbox in the control panel to disable column names. The column in the "Parsed content" section will now be labeled V1. Click on the "Item summary" control as an alternate way to verify that the data is loaded and parsed accurately. The main content area includes two elements, a selection for the "Row frequency column" and a table of items by Outcomes and Missing (see Figure 5). The "Row frequency column" selection is used when you have already reduced your data to unique rows and row counts. The example data set LSAT6 is in this format. For our current data set, leave "Row frequency column" set to −.

The information conveyed in the item table should rouse suspicion. There is only 1 row (or 1 item) with 721 outcomes. What happened is that the parsing is still not working and all the items are treated as a single column. For example, the first row "0 1 0 1 1 1 1 1 1 1 0 0" is not treated as 12 separate numbers but as a single uninterpreted unit. To fix the parsing, we need to select Whitespace as the Separator in the control panel. With this last error corrected, the table is updated with 12 items labeled V1, V2, . . . , V12 and all with 2 outcomes. With the data correctly loaded, click on the "Outcomes" tab on the top bar.

The control panel on the "Outcomes" tab packs a lot of functionality (Figure 6). The first two selectors are mutually exclusive and permit working with all items having the same outcomes or with specific items, respectively. The outcome set "V1" is currently selected as seen in the control panel on the left side. In these data, all items have the same possible outcomes (0 or 1). Therefore, there is only one outcome set. The name "V1" does not just refer to the item "V1" but to all items, because all items have the same outcomes.

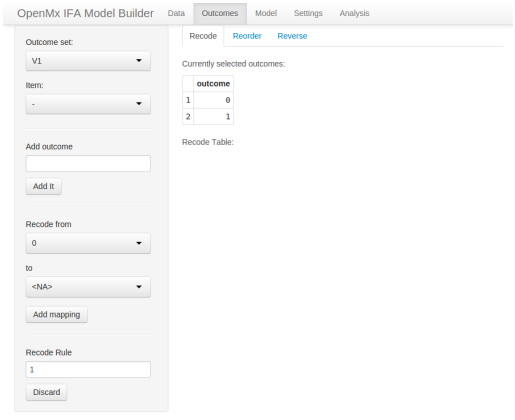


Figure 6: The Outcomes tab without any recoding rules.

For clarity, it is often helpful to rename outcomes. The "Recode from" selector should have "0" selected. Change the to selector to <Rename>, enter "incorrect" for the "New name" value, and click the "Add mapping" button. This will create a recoding rule that will show up in the "Recode Table" output (Figure 7). Similarly, rename the "1" outcome to "correct" and again

[1] It may read "Choose CSV File'." The exact appearance may differ on your system.

[2] We are aware that these screenshots are illegible when printed on paper. Inspect them using magnification on your computer.

click the "Add mapping" button. At this point, you should have 2 rules in the "Recode Table" output.

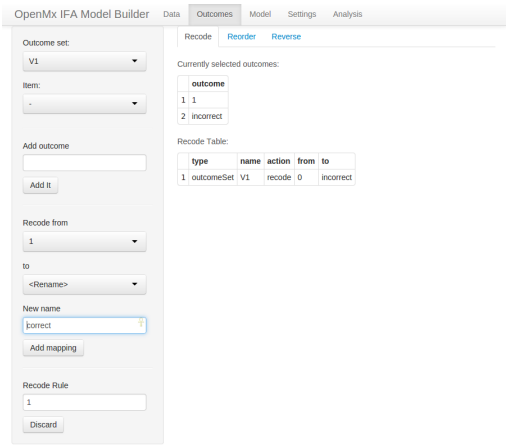


Figure 7: The outcome "0" is renamed to "incorrect" and we are poised to rename "1" to "correct." In a moment, we will click the "Add mapping" button.

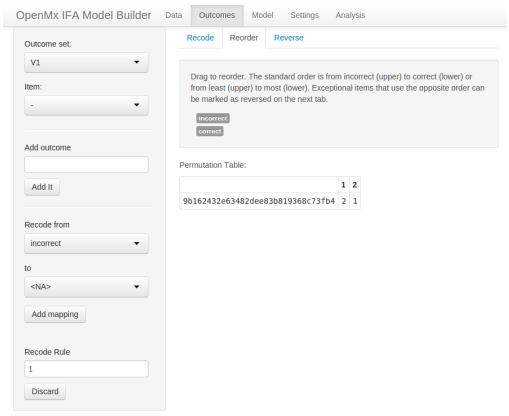


Figure 8: The outcome reorder tool with 1 reordering rule.

Click on the "Reorder" tab. Here you will find the outcomes sorted in lexical order. Drag one of the outcomes to reverse the order (as in Figure 8). Similar to the operation of renaming outcomes, this reordering step is not strictly necessary but is often helpful to keep things straight in our minds. With dichotomous outcomes, there are not that many ways that things can go wrong. However, it is good practice to use self-explanatory labels and standardized ordering. This is especially true when there are more than 2 outcomes to worry about.

We will not use the "Reverse" tab and other control panel elements in the present example. In survey design, it is common for outcomes to have a canonical order with some items reverse scored. The "Reverse" tab is used to reverse the outcome order of some subset of items without dragging the outcomes around with the "Reorder" tool. This can save a lot of dragging when there are more than 2 outcomes. The "Add outcome" tool permits the addition of outcomes that are not represented in the data. This might happen when a measure is in development and we are fitting a preliminary model just to get a vague idea of how the item is working. To fit an item that lacks data on some outcomes, it is usually necessary to use the nominal response model with a less than full rank parameterization (similar to Thissen et al., 1989). In addition to renaming, the recode mappings tool can merge or collapse outcomes. For example, we might have an outcome set consisting of "Agree," "Agree somewhat," "Not sure," "Disagree somewhat," and "Disagree." It is straightforward to merge "Agree somewhat" to "Agree" and "Disagree somewhat" to "Disagree," resulting in only 3 outcomes. Of course, it is not always obvious which outcomes to merge. The recode tool can also recode an outcome to <NA>, which causes that outcome to be interpreted as missing. Finally, the "Discard" button at the bottom of the control panel allows us to discard any rule that we created. So feel free to experiment.

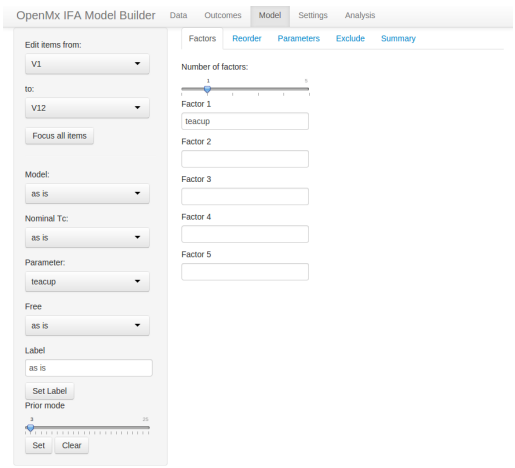


Figure 9: Configuration of latent factors.

Click on the "Model" tab on the top bar. The first decision we need to make is how many latent factors to include in our model (Figure 9). If we are not sure, a good starting point is 1. By default, the first latent factor is called teacup. In case there was any doubt, "teacup" is not a very good name for a latent trait. We deliberately picked ridiculous factors names to encourage users to pick names that make sense in the context of the data under analysis. For example, a good factor name for a math test might be math. If you cannot think of a good factor name, you could use "latent trait," but this name only works well for a single factor model. You really should make an effort to think of descriptive names before you start using trait1, trait2, etc. If you are not sure how many

factors to use then use 1 for now. We will revisit this question later.

The "Reorder" tab allows you to change the order of your items. This can be helpful because of the way that item model and parameter editing works. To get familiar with how item editing works, click on the "Parameters" tab. The main content area of the "Parameters" tab contains a lot of information. The first thing to notice is that all of the tables have the same column labels. Each item is assigned to a column. Using the control panel, we will focus on only a subset of items. Change the first selector "Edit items from:" from V1 to V7. This will hide the first 6 items, making the tables in the main content area look more manageable (Figure 10). The first two selectors facilitate item range selection. To reveal all items again, use the "Focus all items" button. Item selection is important to understand because the remainder of the controls in the control panel operate on only the selected (visible) items.

With only items V7 to V12 visible, just to demonstrate how it is done, let us place an equality constraint on the slope or latent factor loading. Type "slope" into the Label textbox and click the "Set Label" button. The label slope should appear in all columns of the first row of the Labels table in the main content area. Now let us switch to the first 6 items. This can be accomplished in a variety of ways. One way is to change the first selector from V7 to V6 and the second selector from V12 to V1.

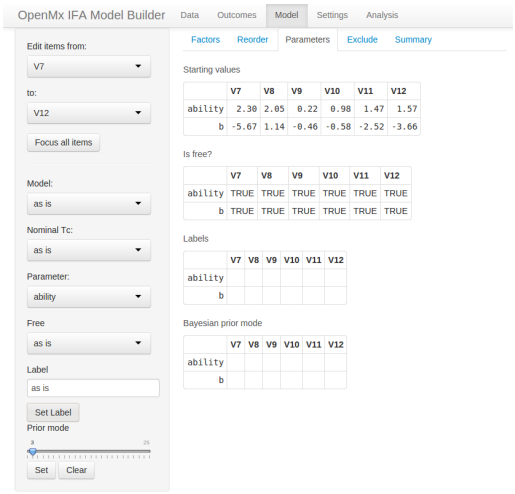


Figure 10: Editing the models and parameters of a subset of items.

With only items V1 to V6 visible, select drm from the "Model" selector. The value drm is an abbreviation for the 4PL dichotomous response model (Loken and Rulison, 2010), which has four parameters when there is one factor. The g and u rows should appear in all of the tables in the main content area. Parameter g is the pseudo-guessing lower bound and u is the upper bound. The upper bound has been used to better model high ability examinees in a Computerized Adaptive Testing context (Magis, 2013). Even high ability examinees may occasionally miss an easy item. Here we will fix the upper bound to 1 (meaning that an examinee with sufficiently high ability will never answer incorrectly). Since the bound parameters are expressed on a logit scale, we will use $\text{logit}(1)$. Select u from the "Parameter" selector and Inf from the "Free" selector (since $\text{logit}(1) = \text{inf}$). The row of "Starting values" for u should change to Inf and the corresponding "Is free" row should change from TRUE to FALSE. With this constraint, the 4PL dichotomous response model is equivalent to the 3PL model (Birnbaum, 1968).

The pseudo-guessing lower bound g represents the chance that an examinee will get the item correct by guessing. Typically, the expected guessed-correct probability for a 3-alternative item is $\frac{1}{3}$ and $\frac{1}{n}$ for an n-alternative item. Estimating the lower bound from data without telling the model a priori how many alternatives were presented typically requires much more data than is required to estimate other kinds of parameters. This is especially true for easy items because few participants will need to guess. It could be argued that easy items should have the lower bound set to a probability of zero. However, in an item set with some lower bounds fixed to zero and some free, the items with the lower bounds fixed to zero will provide more information than the items that take the chance of guessing into account. Therefore, we suggest fixing the lower bound to $\frac{1}{n}$ for an n-alternative item when estimation of the lower bound is not of interest.

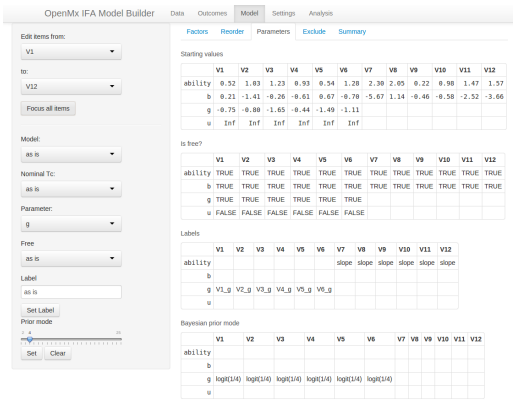


Figure 11: Item tables after setting up our model.

As a compromise between fixing and freeing, a Bayesian prior can be used with the mode of the prior set to the expected probability. While some researchers are uneasy about the use of priors (Gelman, 2008), the practice is not new (e.g, Baker and Kim, 2004, Chapter 7). The prior ensures that a parameter will converge even when there is not enough data to estimate it, but at the same time,

the model retains some flexibility to adapt to unexpected data. To set a prior, drag the "Prior mode" slider and click the "Set" button. Let us imagine that these items had 4 alternatives. Select g from the Parameter selector, move the "Prior mode" slider to 4, and click the nearby Set button. Two tables will change. Each cell of the g row of the Labels table will be assigned a unique label. This is necessary because Bayesian priors are associated with labels, not with particular parameters. In addition, the "Bayesian prior mode" table will show logit$(1/4)$ in the g row. The logit usage reflects that the parameter is estimated on the logit scale, but it is much easier for humans to understand a probability expressed as a fraction rather than raw logit units.

We will not use the "Nominal Tc" selector for these data. "Nominal Tc" only applies to items with more than 2 possible outcomes when using the nominal response model (Thissen et al., 2010). Before proceeding, go ahead and click the "Focus all items" button. Your screen should look like Figure 11, except for different starting values. Click on the "Exclude" tab. This is an easy way to exclude some of the items from analysis. Finally, click on the "Summary" tab. Here you will find a summary of your model settings. Note that the number of outcomes may differ from the number of outcomes reported in the summary table found on under the "Data" top bar page due to recoding.

We are done setting up our model. Before proceeding, it would be wise to save our model configuration so we can come back at a later time and make small adjustments without going through the whole exhaustive process again. Click Settings on the top bar. In the main content area, you will find a preview of what the settings file will look like. Click the "Download" button and move the file to a suitable location on your computer.

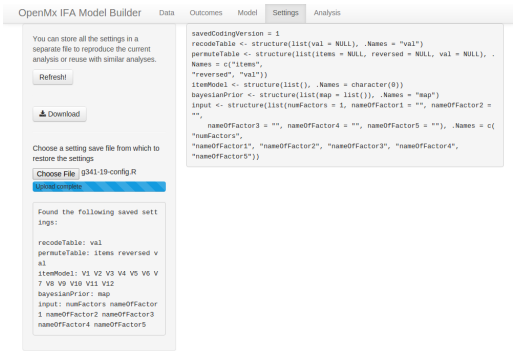


Figure 12: Restoring the settings.

To verify that you can reliably restore the saved settings, open a new browser tab to the same address by pasting the URL address from the current tab (without closing the current one). You should get a screen like Figure 3. Again go through the procedure of loading the data (Figures 4 and 5). Once your data is loaded, click Settings on the top bar and load the file that you recently saved. If all goes well, you should see a screen similar to Figure 12. Go ahead and look back through the editors under the Outcomes and Model sections of the top bar. You should find all your hard work faithfully preserved. Now you can close either of the 2 browser tabs that you have open. They both have the same status.

With our model set up and saved, click Analysis on the top bar. This screen looks and functions similarly to the Settings screen. However, the control panel offers a few options specific to code generation. The "Functional form for dichotomous bound prior density" selector chooses the distributional form of the Bayesian prior. Logit-normal is the more recent scheme (Cai et al., 2011). The "Information matrix method" control is set to Oakes by default. In a simulation study included with **OpenMx**, the Oakes method (Oakes, 1999) exhibited accuracy comparable to central difference with Richardson extrapolation and time linear in the number of parameters. Click the "Download" button and save the Rmarkdown code. The Rmarkdown file and your data need to be in the same directory. Either move the Rmarkdown file to your data directory, or alternately, you can specify a full path in the read.csv statement (lines 16–17). Open the file in **RStudio** and click the "Knit HTML" button.

```
1  ---
2  title: "g341-19"
3  date: "14-Nov-2014"
4  output: html_document
5  ---
6
7  ```{r}
8  options(width = 120, scipen = 2, digits = 2)
9  suppressPackageStartupMessages(library(OpenMx))
10 suppressPackageStartupMessages(library(rpf))
11 suppressPackageStartupMessages(library(ifaTools))
12 library(xtable)
13 options(xtable.type = 'html')
14
15 # Adjust the path in the next statement to load your data
16 data <- read.csv(file = 'g341-19.csv', header = FALSE, sep = ' ',
17   stringsAsFactors = FALSE, check.names = FALSE)
```

```
colnames(data) <- mxMakeNames(colnames(data), unique = TRUE)

factors <- "ability"
numFactors <- length(factors)
spec <- list()
spec[1:6] <- rpf.drm(factors = numFactors)
spec[7:12] <- rpf.grm(factors = numFactors, outcomes = 2)
names(spec) <- c("V1", "V2", "V3", "V4", "V5", "V6", "V7", "V8", "V9", "V10",
  "V11", "V12")

missingColumns <- which(is.na(match(names(spec), colnames(data))))
if (length(missingColumns)) {
  stop(paste('Columns missing in the data:',
    omxQuotes(names(spec)[missingColumns])))
}

data[names(spec)] <- mxFactor(data[names(spec)], levels = 0:1,
  labels = c("incorrect", "correct"))

set.seed(1)   # uncomment to get the same starting values every time
startingValues <- mxSimplify2Array(lapply(spec, rpf.rparam))
rownames(startingValues) <- paste0('p', 1:nrow(startingValues))
rownames(startingValues)[1:numFactors] <- factors

imat <- mxMatrix(name = 'item', values = startingValues,
  free = !is.na(startingValues))
imat$free['p4',1:6] <- FALSE
imat$values['p4',1:6] <- Inf
imat$labels['ability',7:12] <- 'slope'
imat$labels['p3',1:1] <- 'V1_g'
imat$labels['p3',2:2] <- 'V2_g'
imat$labels['p3',3:3] <- 'V3_g'
imat$labels['p3',4:4] <- 'V4_g'
imat$labels['p3',5:5] <- 'V5_g'
imat$labels['p3',6:6] <- 'V6_g'
hasLabel <- !is.na(imat$labels)
for (lab1 in unique(imat$labels[hasLabel])) {
  imat$values[hasLabel & imat$labels == lab1] <-
    sample(imat$values[hasLabel & imat$labels == lab1], 1)
}
data <- compressDataFrame(data)
itemModel <- mxModel(model = 'itemModel', imat,
  mxData(observed = data, type = 'raw',
    numObs = sum(data[['freq']]), sort = FALSE),
  mxExpectationBA81(ItemSpec = spec, weightColumn = 'freq'),
  mxFitFunctionML())

priorLabels <- c("V1_g", "V2_g", "V3_g", "V4_g", "V5_g", "V6_g")
priorParam <- mxMatrix(name = 'priorParam', nrow = 1,
  ncol = length(priorLabels), free = TRUE, labels = priorLabels)
priorParam$values <- imat$values[ match(priorParam$labels, imat$labels) ]
priorMode <- c(priorParam$values)
priorMode[1:6] <- logit(1/4)
priorModel <- univariatePrior('logit-norm', priorLabels, priorMode)
container <- mxModel(model = 'container', itemModel, priorModel,
  mxFitFunctionMultigroup(groups = c('itemModel.fitfunction',
                                     'univariatePrior.fitfunction')))

emStep <- mxComputeEM('itemModel.expectation', 'scores',
  mxComputeNewtonRaphson(), verbose = 2L,
  information = 'oakes1999', infoArgs = list(fitfunction = 'fitfunction'))
computePlan <- mxComputeSequence(list(EM = emStep,
         HQ = mxComputeHessianQuality(),
         SE = mxComputeStandardError()))
```

```
82
83 m1Fit <- mxRun(container)
84
85 m1Grp <- as.IFAgroup(m1Fit$itemModel, minItemsPerScore = 1L)
86 ```
```

The details of the generated report are likely to evolve. There may be some differences between the generated code in this article and the most recent version, but there should be a correspondence between the basic elements. The first chunk of code builds the model and optimizes it. We adjust the output of long lines and numbers (line 8) and load packages (lines 9–13). The data set is loaded (Figure 5) in lines 16–17. Latent factors are configured (Figure 9) in lines 20–24. We strongly encourage the use of informative column (item) names, but line 18 will take care of assigning syntactically valid column names if informative names are not available. The script is crafted such that it can work on other data sets as long as the required columns are found (line 28). `mxFactor` does the recoding and reordering (Figures 6–8). `mxFactor` offers a number of important safety and convenience features beyond what is available from `factor` or `ordered` (line 34). The item `mxMatrix` (line 43) contains most of the information in the item tables (Figure 11). Everything goes into the `container` model (line 72). The model is optimized (line 83). Since we did not disable "`Show model fitting progress`" (reflected by `verbose = 2L` at line 77), we obtain some diagnostics upon knitting the `Rmarkdown` to HTML.

```
87 [0] ComputeEM: Welcome, tolerance=1e-09 accel=varadhan2008 info=2
88 [0] ComputeEM: msteps 2 initial fit 37185.0001
89 [0] ComputeEM[2]: msteps 11 fit 34167.9816 rel change 0.0882995805
90 [0] ComputeEM[3]: msteps 5 fit 33699.978 rel change 0.0138873556
91 [0] ComputeEM[4]: msteps 14 fit 33549.9723 rel change 0.00447111437
92 [0] ComputeEM[5]: msteps 5 fit 33455.9478 rel change 0.00281039684
93 [0] ComputeEM[6]: msteps 3 fit 33454.4705 rel change 4.41596231e-05
94 [0] ComputeEM[7]: msteps 3 fit 33453.8021 rel change 1.99793343e-05
95 [0] ComputeEM[8]: msteps 3 fit 33453.2067 rel change 1.77968988e-05
96 [0] ComputeEM[9]: msteps 2 fit 33453.2062 rel change 1.57420931e-08
97 [0] ComputeEM[10]: msteps 2 fit 33453.206 rel change 5.03007605e-09
98 [0] ComputeEM[11]: msteps 2 fit 33453.2059 rel change 2.89615064e-09
99 [0] ComputeEM[12]: msteps 2 fit 33453.2059 rel change 6.61823582e-10

100 [0] ComputeEM: cycles 12/500 total mstep 54 fit 33453.205893
101 [0] ComputeEM: Oakes1999 method=simple perturbation=0.001000
102 [0] ComputeEM: deriv of gradient for 0
103 [0] ComputeEM: deriv of gradient for 1
104 [0] ...
105 [0] ComputeEM: deriv of gradient for 24
```

Given that the starting values are random, the initial fit and trajectory (lines 88–99) should differ when you try optimizing the same model but the optimum (line 100) should be the same to within 10^{-2}. If you do not reach the same solution, try again with different starting values. At the time of writing, optimization is faster on multicore CPUs running on operating systems other than Microsoft Windows. As soon as Windows supports OpenMP then we expect performance differences between operating systems to narrow.

The function `as.IFAgroup` (line 85) is a bridge between **OpenMx** and **rpf** (Pritikin, 2015b). The **rpf** name is an acronym for response probability function and contains many IFA-specific diagnostic functions. In addition, **rpf** can be used to analyze IFA models optimized by packages other than **OpenMx**. This modularity facilitates the comparison of parameter estimates between packages. While most of the code discussed so far is related to **OpenMx**, the remainder of this report will mostly involve **rpf**.

```
106 An item factor model was fit with `r length(factors)`
107 factors (`r factors`), -2LL = $`r m1Fit$output$fit`$.
108 The condition number of the information
109 matrix was `r m1Fit$output$conditionNumber`.
```

This is a boilerplate report of model fit. It renders as, "An item factor model was fit with 1 factors (ability), −2LL = 33453.21. The condition number of the information matrix was 70.91." It is not really feasible to generate a complete Results section because there are always considerations idiosyncratic to a particular project that dictate how the Results section should best unfold. However, it is likely that some additional boilerplate reporting will be added to the model builder app in a future release.

Although IFA models consider an examinee's response pattern as the unit of analysis, the sum-score is often of chief practical importance. For example, students taking the Standardized Aptitude

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
V2	0.1										
V3	−0.0	−2.6									
V4	1.7	−0.1	1.2								
V5	−0.6	−0.9	−1.3	5.1							
V6	0.2	0.1	−0.3	0.3	−1.1						
V7	0.1	6.4	0.6	2.4	0.3	5.0					
V8	−0.5	−0.3	−0.7	−4.0	−0.2	0.1	3.9				
V9	3.6	10.7	1.1	2.6	1.8	5.8	37.1	0.3			
V10	−2.0	9.1	0.3	−0.3	0.1	−0.2	10.2	−0.5	16.2		
V11	−1.0	−11.5	−2.6	−1.1	−0.6	−1.4	−1.9	−4.8	−0.5	−0.7	
V12	−0.1	−1.7	3.9	−2.9	−1.9	−0.7	0.8	−2.0	−0.1	0.6	−7.1

Table 1: Log p-value of local dependence between item pairs.

Test for college admission only receive their sum-score and do not even know which items they answered correctly or incorrectly (unless they earned the maximum sum-score). The observation that the sum-score is important has lead to the development of a family of diagnostic tests that examine how well an IFA model predicts sum-scores.

```
```{r,fig.height=2}
got <- sumScoreEAPTest(m1Grp)
df <- data.frame(score = as.numeric(names(got[['observed']])),
  expected = got[['expected']], observed = got[['observed']])
df <- melt(df, id = 'score', variable.name = 'source',
  value.name = 'n')
ggplot(df, aes(x = score, y = n, color = source)) + geom_line()
```
```

The first plot provides an overview of how all the items work together to predict sum-scores (Figure 13). `sumScoreEAPTest` also conducts a statistical test to produce a p-value, but this is not reported here because the test is fairly new and the meaning of the test has not yet been well established (Li and Cai, 2012). However, it is still worth looking at this plot because you might notice something that is obviously wrong with the model (i.e., if the curves mismatch drastically).

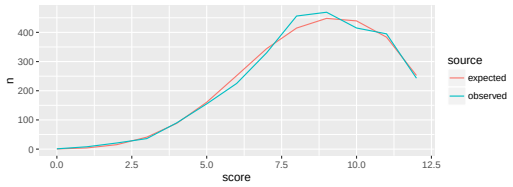


Figure 13: Comparison of the predicted and observed sum-scores.

```
```{r,results='asis'}
ct <- ChenThissen1997(m1Grp)
print(xtable(ct$pval,
  paste('Log p-value of local dependence between item pairs.')))
```
```

A test of local dependence is important to examine, as it is an assumption of IFA (Yen, 1993). Table 1 exhibits the log p-value of the null hypothesis that there is no local dependence between item pairs. Since $\log(.01) \approx -4.6$, any absolute magnitude greater than 4.6 can be interpreted as rejecting the null hypothesis at the .01 level. The sign of each p-value is determined by *ordinal gamma*, a measure of association (Agresti, 1990). Positive numbers indicate more correlation than expected. These are cause for concern and suggest local dependence (Chen and Thissen, 1997). Negative numbers indicate less correlation than expected. Table 1 is also a good example of a weakness of comparing expected and observed frequencies: all you can know is that *something* is suboptimal, but not specifically what. The local dependence is most severe between item pairs V7/V9, V9/V10, and V2/V9. Item pair V2/V11 also has a large magnitude value, but this is less of a concern because the sign is negative. Unfortunately, this diagnostic only reveals potential deficiencies, but does not suggest how to address them. Improvement of the model (or the items) often requires some guesswork and trial-and-error.

```
```{r,results='asis'}
sfit <- SitemFit(m1Grp)
tbl <- t(sapply(sfit, function(r)
  c(n = r$n, df = r$df, stat = r$statistic, pval = r$pval)))
```
```

	n	df	statistic	log p-value
V1	2844	8	6.58	−0.54
V2	2844	8	7.30	−0.68
V3	2844	8	7.17	−0.66
V4	2844	8	10.12	−1.36
V5	2844	8	19.00	−4.21
V6	2844	8	8.50	−0.95
V7	2844	9	33.45	−9.10
V8	2844	9	5.48	−0.24
V9	2844	9	34.42	−9.49
V10	2844	10	12.61	−1.40
V11	2844	8	43.06	−13.97
V12	2844	8	20.20	−4.64

Table 2: Sum-score item-wise fit.

```
print(xtable(tbl, paste0('Sum-score item-wise fit.'))
```

Sum-score item fit is another tool for model assessment (Orlando and Thissen, 2000; Kang and Chen, 2008). Each item is tested against the null hypothesis that the other items accurately predict the outcome of the item of interest (Table 2). Again p-values are in log units so a magnitude larger than 4.6 is significant at the .01 level. In contrast to the test for local dependence, the sign of the p-value does not mean anything special here. The column n is included for data sets with missingness. When there is missingness, it can be advantageous to exclude the item with the most missing values to increase the sample size of the test. Refer to the `SitemFit` help for details on the options for missing data.

```
```{r,fig.height=2}
map1 <- itemResponseMap(m1Grp, factor = 1)
ggplot(map1, aes(x = score, y = item, label = outcome)) +
  geom_text(size = 4, position = position_jitter(h = .25))
```
```

An item response map is a crude tool for diagnosing certain model misspecifications (Figure 14). Each outcome is assigned the average latent score of all the examinees who picked that outcome. Usually we advocate the use of the actual outcome labels (e.g., "incorrect" and "correct") instead of numbers. For this plot, however, we make an exception because the numbers make it easy to inspect whether the outcomes are in ascending order. If descending order is observed then it is worth checking whether the item needs to be reverse scored or to consider whether the item was misinterpreted by some examinees. If the response data were manually collected then the data entry process should also be checked for errors.

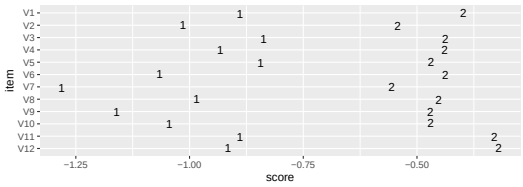


Figure 14: Item outcome by average latent score.

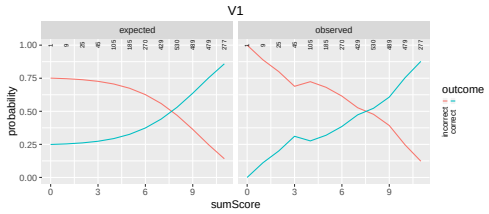


Figure 15: Expected and observed outcome by sum-score.

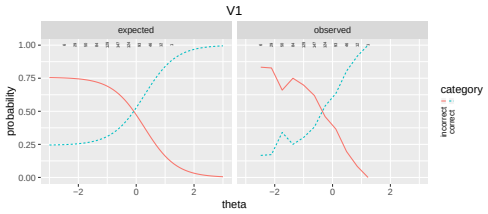


Figure 16: Expected and observed outcome by latent score.

```
```{r,fig.height=3}
pl <- lapply(names(sfit), function(item) { SitemPlot(sfit, item) })
for (px in 1:length(pl)) {
```
```

```
    print(pl[[px]])
}
```

Two approaches are available to plot response probability functions against a latent trait. The same ingredients that go into the production of Table 2 can also be plotted (Figure 15). A similar plot can be obtained by plotting the outcomes probabilities against the latent trait. This is known as an item characteristic curve plot (Figure 16). The main advantage of `SitemPlot` over `iccPlot` is that `SitemPlot` is one-dimensional regardless of the number of latent factors. With `iccPlot`, you must pick a basis vector in the latent space. The tiny numbers across the probability = 1 line of Figures 15 and 16 are the sample size at that point on the x axis.

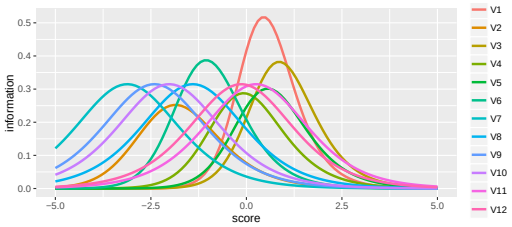


Figure 17: Item information by latent score.

```
basis <- rep(0, length(factors))
basis[1] <- 1
plotInformation(m1Grp, width = 5, basis = basis)
```

Figure 17 exhibits item information by latent score. Similar to `iccPlot`, this plot requires the selection of a basis vector when there is more than 1 latent factor. Notice that items V7-V12 peak at the same height (near 0.31). This is due to our equality constraint on the slope or factor loading on these items. By placing this constraint, we assume a priori that each of these items contributes exactly the same amount of information.

```
```{r}
summary(m1Fit)
```
```

```
Summary of container

free parameters:
                   name          matrix     row col Estimate Std.Error
1   itemModel.item[1,1] itemModel.item ability  V1     1.82     0.278
2   itemModel.item[2,1] itemModel.item      p2  V1    -0.51     0.230
3                  V1_g itemModel.item      p3  V1    -1.14     0.208
4   itemModel.item[1,2] itemModel.item ability  V2     1.24     0.119
5   itemModel.item[2,2] itemModel.item      p2  V2     2.58     0.140
6                  V2_g itemModel.item      p3  V2    -1.27     0.337
7   itemModel.item[1,3] itemModel.item ability  V3     1.56     0.261
8   itemModel.item[2,3] itemModel.item      p2  V3    -1.03     0.272
9                  V3_g itemModel.item      p3  V3    -1.16     0.192
10  itemModel.item[1,4] itemModel.item ability  V4     1.36     0.161
11  itemModel.item[2,4] itemModel.item      p2  V4     0.41     0.158
12                 V4_g itemModel.item      p3  V4    -1.10     0.277
13  itemModel.item[1,5] itemModel.item ability  V5     1.41     0.196
14  itemModel.item[2,5] itemModel.item      p2  V5    -0.47     0.203
15                 V5_g itemModel.item      p3  V5    -1.03     0.203
16  itemModel.item[1,6] itemModel.item ability  V6     1.50     0.130
17  itemModel.item[2,6] itemModel.item      p2  V6     1.84     0.119
18                 V6_g itemModel.item      p3  V6    -1.43     0.317
19                slope itemModel.item ability  V7     1.12     0.037
20  itemModel.item[2,7] itemModel.item      p2  V7     3.50     0.097
21  itemModel.item[2,8] itemModel.item      p2  V8     1.57     0.056
22  itemModel.item[2,9] itemModel.item      p2  V9     2.70     0.075
23 itemModel.item[2,10] itemModel.item      p2 V10     2.27     0.066
24 itemModel.item[2,11] itemModel.item      p2 V11    -0.28     0.047
25 itemModel.item[2,12] itemModel.item      p2 V12     0.15     0.047

observed statistics:  720
estimated parameters:  25
degrees of freedom:  695
```

```
fit value ( -2lnL units ):  33453
number of observations:  2844
Information Criteria:
      |  df Penalty  |  Parameters Penalty  |  Sample-Size Adjusted
AIC:          32063                 33503                      NA
BIC:          27926                 33652                   33573
To get additional fit indices, see help(mxRefModels)
timestamp: 2016-02-24 10:14:59
Wall clock time (HH:MM:SS.hh): 00:00:02.72
OpenMx version number: 2.3.1.254
Need help?  See help(mxSummary)
```

Exhibited above is the **OpenMx** provided summary of model fit. IFA models are exponential family models so we obtain AIC and BIC. More fit statistics are available if we provide the saturated and independence reference models. Reference models will be requested in our next example.

Polytomous data

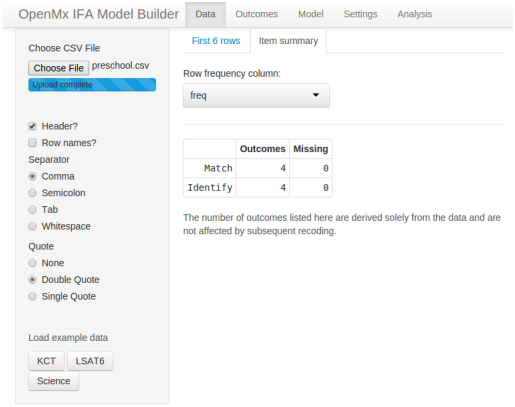


Figure 18: Data with a row frequency column.

Since many things are common between dichotomous and polytomous items, we will move quickly through the process of model set up and result interpretation. Click on the "`Choose File`" button and select `preschool.csv`, a data set from Thissen and Steinberg (1988) available in the **ifaTools** package. Click the "`Row names?`" checkbox in the control panel to disable row names. The format of these data are closer to what is expected by default than our first example so less fiddling is required. Click on the "`Item summary`" tab. Here it appears that there are 3 items, but the `freq` column is not an item. `freq` indicates how many times a row appeared in the original data. These data are compressed; only unique rows are provided with frequency counts. To instruct the model builder to interpret the `freq` column as frequency counts, select `freq` from the "`Row frequency column`" selector (Figure 18).

This data set is from a preschool test of numerical knowledge. Each item is actually a combination of 2 dichotomous items. Similar questions were asked regarding the number 3 and the number 4 and the pattern of responses mapped to an outcome code. The outcomes should be renamed with the recoding tool under the "`Outcomes`" tab on the top bar (recall Figure 7). Outcomes 0, 1, 2, and 3 should be renamed to "neither," "3 only," "4 only," and "both correct," respectively, using the "`Recode`" tab under the `Outcomes` top bar page. After renaming, reorder the items into the correct order (Figure 19).

Click `Model` on the top bar. On the "`Factors`" tab, we will name the single latent factor "math." Switch to the `Parameters` tab. Here we select `nrm` from the "`Model`" selector. The acronym "nrm" stands for the nominal response model (Thissen et al., 2010). This parameterization of the nominal model can accommodate basis matrices T_a and T_c to customize the meaning of the slope and intercept coefficients, respectively. In principle, the basis matrices can take any pattern, but the model builder app is limited to a Fourier basis (a.k.a. trend basis) for the T_a matrix and a small number of options for the T_c matrix.

With T_a set to the trend basis, we cannot free both `math` and `alf1` because they have the same effect on the model and would cause the model to be unidentified. Fix `alf1` to 1. Select `alf1` from the "`Parameter`" selector and select 1 from the "`Free`" selector. Since we have worked with this data set already, we know a few things that can give us a more parsimonious model. The `alf2` parameters can be set equal since both items exhibit poor discrimination between `neither`, `3 only`, and `4 only` but good discrimination between these outcomes and `both correct`. Select `alf2` with the "`Parameter`" selector and set the label to `eq1`. Since both items are equally difficult, we can equate `gam1`. Select `gam1` with the "`Parameter`" selector and set the label to `eq2`. To avoid overfitting with the highest frequency basis vector, fix `gam3` to 0. Select `gam3` with the `Parameter` selector and select `0` with the `Free` selector. Figure 20 exhibits the final parameter settings.

Click `Analysis` on the top bar. Ensure that "`Fit reference models`" is selected, and download the analysis script. The `Rmarkdown` file and your data need to be in the same directory. Either move the `Rmarkdown` file to your data directory, or alternately, you can specify a full path in the read.csv

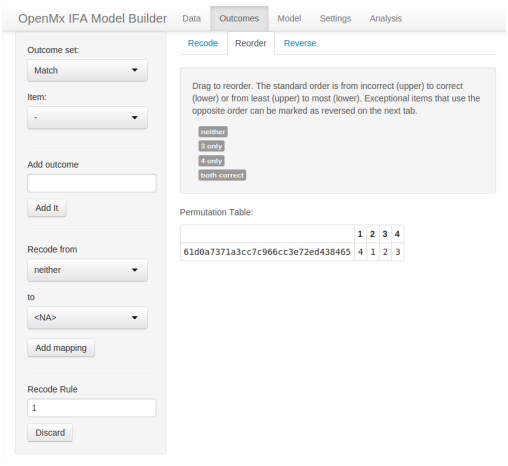


Figure 19: Outcomes renamed and reordered.

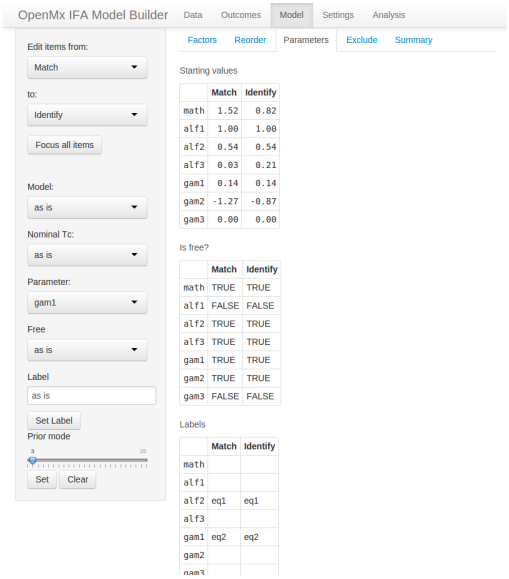


Figure 20: Item model and parameter configuration with equality constraints.

statement (line 162). Open the file in **RStudio** and click the "`Knit HTML`" button. Although this is a simple model, it can take almost 100 E-M cycles to converge. Therefore, we omit reproduction of the diagnostic output issued during model fit.

```
---
title: "preschool"
date: "18-Nov-2014"
output: html_document
---

```{r}
options(width = 120, scipen = 2, digits = 2)
suppressPackageStartupMessages(library(OpenMx))
suppressPackageStartupMessages(library(rpf))
suppressPackageStartupMessages(library(ifaTools))
library(xtable)
options(xtable.type = 'html')

# Adjust the path in the next statement to load your data
data <- read.csv(file = 'preschool.csv', stringsAsFactors = FALSE,
  check.names = FALSE)
colnames(data) <- mxMakeNames(colnames(data), unique = TRUE)
data[['freq']] <- as.numeric(data[['freq']])

factors <- "math"
numFactors <- length(factors)
spec <- list()
spec[1:2] <- rpf.nrm(factors = numFactors, outcomes = 4,
  T.a = 'trend', T.c = 'trend')
names(spec) <- c("Match", "Identify")

missingColumns <- which(is.na(match(names(spec), colnames(data))))
if (length(missingColumns)) {
  stop(paste('Columns missing in the data:',
       omxQuotes(names(spec)[missingColumns])))
}

data[names(spec)] <- mxFactor(data[names(spec)], levels = 0:3,
  labels = c("neither", "3 only", "4 only", "both correct"))

```
```

	n	df	statistic	log p-value
Match	592	7	9.46	−1.51
Identify	592	7	9.33	−1.47

Table 3: Sum-score item-wise fit.

```
set.seed(1)   # uncomment to get the same starting values every time
startingValues <- mxSimplify2Array(lapply(spec, rpf.rparam))
rownames(startingValues) <- paste0('p', 1:nrow(startingValues))
rownames(startingValues)[1:numFactors] <- factors

imat <- mxMatrix(name = 'item', values = startingValues,
  free = !is.na(startingValues))
imat$free['p2',] <- FALSE
imat$values['p2',1:2] <- 1
imat$free['p7',] <- FALSE
imat$values['p7',1:2] <- 0
imat$labels['p3',] <- 'eq1'
imat$labels['p5',] <- 'eq2'
hasLabel <- !is.na(imat$labels)
for (lab1 in unique(imat$labels[hasLabel])) {
  imat$values[hasLabel & imat$labels == lab1] <-
    sample(imat$values[hasLabel & imat$labels == lab1], 1)
}
itemModel <- mxModel(model = 'itemModel', imat,
  mxData(observed = data, type = 'raw', numObs = sum(data[['freq']]),
    sort = FALSE),
  mxExpectationBA81(ItemSpec = spec, weightColumn = 'freq'),
  mxFitFunctionML())

emStep <- mxComputeEM('itemModel.expectation', 'scores',
  mxComputeNewtonRaphson(), verbose = 2L,
  information = 'oakes1999',
  infoArgs = list(fitfunction = 'fitfunction'))
computePlan <- mxComputeSequence(list(emStep,
         mxComputeHessianQuality(),
         mxComputeStandardError()))

m1Fit <- mxRun(mxModel(itemModel, computePlan))

m1Grp <- as.IFAgroup(m1Fit, minItemsPerScore = 1L)
```

Although response pattern frequencies are typically natural numbers, fractional frequencies are not prohibited (line 164). A Fourier basis is used for both nominal model transformation matrices (line 170). Customization is limited in the model builder app, but you can use any matrices by editing the generated code. Starting values must respect equality constraints (line 197). By default **OpenMx**, sorts data prior to optimization. Since our data are already compressed, no benefit would be obtained by sorting (line 202).

```
An item factor model was fit with `r length(factors)`
factors (`r factors`), -2LL = $`r m1Fit$output$fit`$.
The condition number of the information matrix was
`r round(m1Fit$output$conditionNumber)`.
```

The boilerplate renders as, "An item factor model was fit with 1 factors (math), −2LL = 2767.48. The condition number of the information matrix was 85.07." Since we have already seen much of the code to generate model diagnostics, we omit it here.

```
```{r}
summary(m1Fit, refModels = mxRefModels(m1Fit, run = TRUE))
```
```

```
Summary of itemModel

free parameters:
                 name matrix  row      col Estimate Std.Error
1 itemModel.item[1,1]   item math    Match     0.82      0.37
2                 eq1   item   p3    Match    -1.18      0.27
3 itemModel.item[4,1]   item   p4    Match     0.50      0.19
4                 eq2   item   p5    Match     0.18      0.05
5 itemModel.item[6,1]   item   p6    Match    -0.78      0.20
6 itemModel.item[1,2]   item math Identify     0.79      0.35
7 itemModel.item[4,2]   item   p4 Identify    -0.25      0.36
8 itemModel.item[6,2]   item   p6 Identify    -1.40      0.21

observed statistics:  15
estimated parameters:  8
degrees of freedom:  7
fit value ( -2lnL units ):  2767
saturated fit value ( -2lnL units ):  2758
number of observations:  592
chi-square:  X2 ( df=7 ) = 9.2,  p = 0.24
Information Criteria:
      |  df Penalty  |  Parameters Penalty  |  Sample-Size Adjusted
AIC:           2753                  2783                     NA
BIC:           2723                  2819                   2793
CFI: 0.98
TLI: 0.97   (also known as NNFI)
RMSEA:  0.023  [95% CI (0, 0.064)]
Prob(RMSEA <= 0.05): 0.88
timestamp: 2016-02-24 10:15:07
Wall clock time (HH:MM:SS.hh): 00:00:04.55
OpenMx version number: 2.3.1.254
Need help?  See help(mxSummary)
```

Although the outcomes are not strictly ordered for `Identify` in the item outcome map (Figure 21), other measures of model fit look reasonable. The sum-score item fit tests are not statistically significant at the 0.01 level (Table 3). This indicates good item-level fit. Since we requested a saturated and independence model (`mxRefModels`; line 223), CFI (Comparative Fit Index), TLI (Tucker Lewis Index), and RMSEA (Root Mean Square Error of Approximation) are available in the **OpenMx** summary and suggest adequate model fit. These relative indices of fit are appropriate for these data because there are observations for all possible response patterns. However, be forewarned that as the multinomial table becomes more sparse, these indices become inaccurate. For sparse data, a more accurate assessment of model fit is available using other methods (Bartholomew and Tzamourani, 1999; Cai and Hansen, 2013).

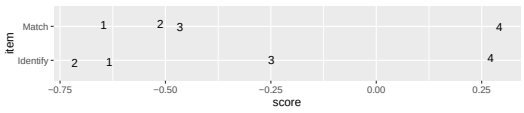


Figure 21: Item outcome by average latent score.

Rasch diagnostics

A Rasch model is obtained when all slope parameters are constrained to be equal and the variance is fixed to 1.0, or equivalently, all slopes are fixed to 1.0 with free variance (Rasch, 1960/1993). If your interest is Rasch models with a single latent factor then you can take advantage of Rasch residual-based fit statistics. Infit and outfit are available from `rpf.1dim.fit`.

Item factor analysis

A common problem is that we do not know how many latent factors to employ to most accurately model our data. Fortunately, there is a method item factor analysis (Bock et al., 1988) analogous to factor analysis of continuous indicators (Lovie and Lovie, 1996). We will employ the likelihood ratio test for inference. The likelihood ratio test is asymptotically consistent for sparse multinomial

```
> container2 <- container
> container2$itemModel$item$labels['ability', ] <- NA
> m3 <- addExploratoryFactors(container2, 0)
> m3 <- mxRun(m3, silent = TRUE)
> mxCompare(m3, m1)
       base comparison ep minus2LL  df   AIC diffLL diffdf       p
1 container1       <NA> 30    33369 690 31989     NA     NA      NA
2 container1 container1 25    33454 695 32064     85      5 7.7e-17
> m4 <- addExploratoryFactors(container2, 1)
> m4 <- mxRun(m4, silent = TRUE)
> mxCompare(m4, m2)
       base comparison ep minus2LL  df   AIC diffLL diffdf     p
1 container2       <NA> 41    33325 679 31967     NA     NA    NA
2 container2 container2 36    33339 684 31971     14      5 0.013
> grid.arrange(plotTwoFactors(m2$itemModel$item$values[1:2, ]) +
+   labs(title = "a."), plotTwoFactors(m4$itemModel$item$values[1:2, ]) +
+   labs(title = "b."), ncol = 2)
```

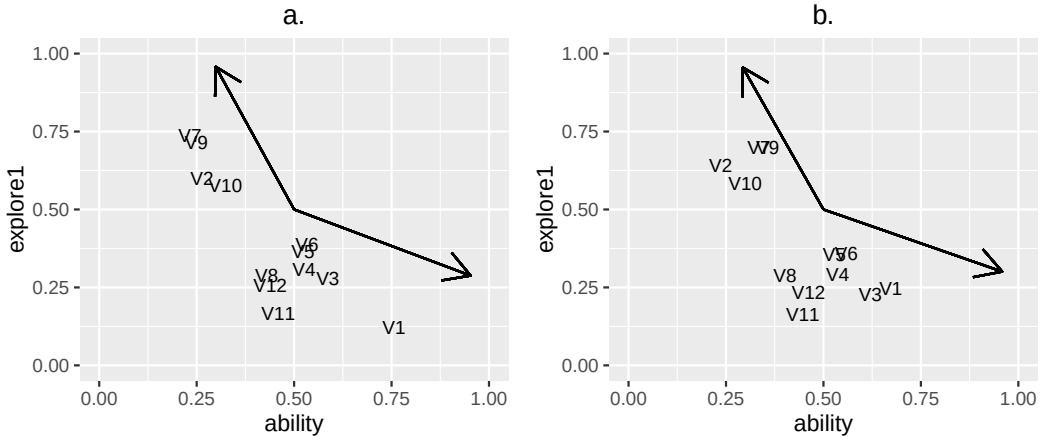


Figure 22: Factor loadings for items with (a) and without (b) the slope constraint. The code for `plotTwoFactors` is given in the Appendix.

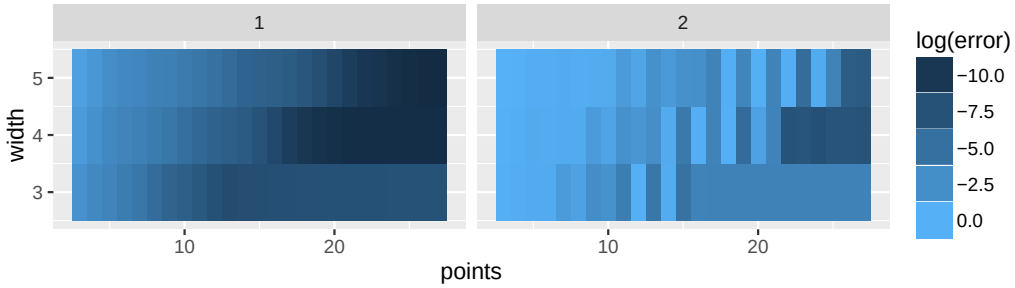


Figure 23: Log Euclidean distance (l^2-norm) of error by quadrature width and number of points for 1 factor (left) and 2 factors (right). A wider width is important to accommodate data that conform less closely to a normal distribution. Even with clean simulated data, a width of 3 is too narrow and interferes with accuracy (both panes). In the 1 factor case (left), at least 21 points are required for high accuracy. For 2 factors (right), at least 23 points are required for a width of 4 and 27 points for a width of 5. The bright strips at even numbers of point (12, 14, 16, etc) indicate that an odd number of points obtain somewhat better accuracy than even numbers of points.

distributions (Haberman, 1977). However, in finite samples, we should not expect that the null distribution is well calibrated. In brief, the p-values should not be taken too seriously.

```
> m1 <- addExploratoryFactors(container, 0)
> m1 <- mxRun(m1, silent = TRUE)
> m2 <- addExploratoryFactors(container, 1)
```

```
> m2 <- mxRun(m2, silent = TRUE)
> mxCompare(m2, m1)
        base comparison ep minus2LL  df   AIC diffLL diffdf       p
1 container2       <NA> 36    33339 684 31971     NA     NA      NA
2 container2 container1 25    33454 695 32064    115     11 1.5e-19
```

Here we find that there is reasonably good support in favor of a two factor solution. However, the slope of items 7-12 are constrained equal. Maybe this constraint was a mistake. It is possible that these items are well modeled by a single factor when all the slopes are freed. We cannot directly compare `m2` against a single factor model without the slope constraint because these models are not nested. However, we can make a number of similar comparisons.

We find that there is a dramatic improvement in fit whether we relax the constraint on items 7-12 or we add another factor. Without knowing more about how the data were collected, parsimony favors a single factor model without constraints on the slopes. We can further check this idea by comparison of two factor models with and without the slope constraint (Figure 22).

A p-value of 0.013 is statistically significant at the customary 0.05 level, but we regard this as non-significant in comparison to the other p-values that are less than 10^{-16}. We conclude that there is no difference between these models. For two factor models, it can be helpful to plot item factor loadings. A varimax rotation eliminates rotational indeterminacy. Promax axes are helpful to illustrate the rough directions of variability (Bock et al., 1988, p. 265). In both plots, the promax axes are separated by an angle close to π radians, suggesting a single latent factor. The slight differences between plots (a) and (b) are probably due to overfitting. More precise p-values could be obtained using Monte Carlo techniques.

Repercussions of the use of numerical quadrature for integration

Recall that the optimization algorithm uses equal interval quadrature to evaluate the integral in Equation 2. It is important to understand how the quadrature grid influences model optimization accuracy and time. Let Q be the number of quadrature points per dimension and Q_{width} be the one-sided width of the quadrature for one dimension. Points X_q are arranged as

$$X_q = Q_{width}(1 - \frac{2q}{Q-1}) \quad \text{for } q \in \{0, \ldots, Q-1\}. \tag{12}$$

Generalization to more dimensions is accomplished by replication of the same 1 dimensional grid along each dimension. For example, a two factor model with 31 points per dimension involve $31^2 = 961$ grid points. Hence, optimization time is exponential in the number of general factors.

Figure 23 exhibits a simulation study of the influence of quadrature on model accuracy. All comparisons are against a 41 point quadrature of width 5.0. Before computing the Euclidean distance (l^2-norm), the slope matrix was converted into factor loadings,

$$\frac{\text{slope}}{\left[1 + \text{rowSums}(\text{slope}^2)\right]^{\frac{1}{2}}}. \tag{13}$$

For two factor models, a varimax rotation was applied to eliminate rotational indeterminacy. The l^2-norm was applied to the resulting slope entries (ignoring intercepts). Each grid area in Figure 23 represents the average of 5 trials with different random starting values.

Item factor analysis with more than two factors requires patience and expertise. Model optimization time becomes an uncomfortable hindrance to experimentation. An optimization algorithm better suited to many latent factors, such as the Metropolis-Hastings Robbins-Monro algorithm (Cai, 2010b), is not yet available in **OpenMx**. The model builder offers as many as five factors because additional factors do not always increase estimation time. Suppose all items load on a general factor. In the special case that each item loads on at most one additional factor, many additional factors will not increase estimation time. One important use for this kind of factor structure is to account for local dependence (DeMars, 2006). For example, a reading comprehension test might have 3-4 items that relate to a single passage. The items within each passage will likely exhibit local dependence. One way to account for this kind of test structure is to add passage specific latent factors. Since the passages are disjoint, all of the passage specific factors will count as a single factor with respect to estimation time (Cai, 2010a).

Discussion

We gave detailed instructions on how to set up IFA models for analysis of both dichotomous and polytomous data using the model builder app. We hope this will ease the learning curve for the construction of IFA models in **OpenMx**. The model builder app offers limited flexibility by design to reduce the number of options for novice users. For example, there is no facility for construction of multiple group models. This may be construed as a disadvantage, but we argue that keeping the app as simple as possible is important for newcomers to IFA. Learning **OpenMx** can be a daunting prospect. **OpenMx**, **rpf**, and **ifaTools** are free software. The source code is available for everybody to view, modify, and use. If you find this software useful, we hope you will cite us in your publications.

Appendix

Factors are plotted in a coordinate system determined by a varimax rotation (line 2). Promax axes are superimposed (line 9).

```
1  plotTwoFactors <- function(slope) {
2    lvm <- varimax(toFactorLoading(slope))$loadings
3    if (any(abs(lvm[lvm < 0]) > .001)) stop("Got negative loadings")
4    lvm[lvm<0] <- 0
5    df <- as.data.frame(lvm[, 1:2])
6    df$name <- rownames(df)
7    pl <- ggplot(df, aes_string(x = rownames(slope)[1],
8      y = rownames(slope)[2], label = "name")) + geom_text(size = 3)
9    pm <- promax(lvm[, 1:2])$rotmat
10   for (dx in 1:ncol(pm)) {
11     d1 <- .5 * pm[, dx] / sqrt(sum(pm[, dx]^2))
12     pl <- pl + geom_segment(x = .5, y = .5, xend = d1[1] + .5,
13       yend = d1[2] + .5, arrow = arrow(length = unit(.5, "cm")))
14   }
15   pl + xlim(0, 1) + ylim(0, 1)
16 }
```

Bibliography

A. Agresti. *Analysis of Categorical Data*. Wiley, New York, 1990. [p192]

J. Allaire, J. McPherson, Y. Xie, H. Wickham, J. Cheng, and J. Allen. *Rmarkdown: Dynamic Documents for R*, 2014. URL http://rmarkdown.rstudio.com. R package version 0.3.8. [p182]

F. B. Baker and S. H. Kim. *Item Response Theory: Parameter Estimation Techniques*. CRC Press, 2nd edition, 2004. [p188]

D. J. Bartholomew and P. Tzamourani. The goodness of fit of latent trait models in attitude measurement. *Sociological Methods & Research*, 27(4):525–546, 1999. [p198]

A. Birnbaum. Some latent trait models and their use in inferring an examinee's ability. In F. M. Lord and M. R. Novick, editors, *Statistical Theories of Mental Test Scores*, pages 397–479. Addison-Wesley, Reading, MA, 1968. [p183, 184, 188]

R. D. Bock and M. Aitkin. Marginal maximum likelihood estimation of item parameters: Application of an EM algorithm. *Psychometrika*, 46:443–459, 1981. [p182]

R. D. Bock, R. Gibbons, and E. Muraki. Full-information item factor analysis. *Applied Psychological Measurement*, 12(3):261–280, 1988. [p198, 200]

L. Cai. A two-tier full-information item factor analysis model with applications. *Psychometrika*, 75(4): 581–612, 2010a. [p200]

L. Cai. High-dimensional exploratory item factor analysis by a Metropolis-Hastings Robbins-Monro algorithm. *Psychometrika*, 75(1):33–57, 2010b. [p182, 183, 200]

L. Cai and M. Hansen. Limited-information goodness-of-fit testing of hierarchical item factor models. *British Journal of Mathematical and Statistical Psychology*, 66(2):245–276, 2013. [p198]

L. Cai, J. S. Yang, and M. Hansen. Generalized full-information item bifactor analysis. *Psychological Methods*, 16(3):221–248, 2011. [p189]

W.-H. Chen and D. Thissen. Local dependence indexes for item pairs using Item Response Theory. *Journal of Educational and Behavioral Statistics*, 22(3):265–289, 1997. [p192]

C. E. DeMars. Application of the bi-factor multidimensional Item Response Theory model to testlet-based tests. *Journal of Educational Measurement*, 43(2):145–168, 2006. [p200]

A. Gelman. Objections to Bayesian statistics. *Bayesian Analysis*, 3(3):445–449, 2008. [p188]

S. J. Haberman. Log-linear models and frequency tables with small expected cell counts. *The Annals of Statistics*, 5(6):1148–1169, 11 1977. doi: 10.1214/aos/1176344001. [p199]

T. Kang and T. T. Chen. Performance of the generalized S-X^2 item fit index for polytomous IRT models. *Journal of Educational Measurement*, 45(4):391–406, 2008. [p193]

D. E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984. [p182]

Z. Li and L. Cai. Summed score likelihood based indices for testing latent variable distribution fit in item response theory. In *Annual International Meeting of the Psychometric Society*, Lincoln, NE, 2012. [p192]

E. Loken and K. L. Rulison. Estimation of a four-parameter item response theory model. *British Journal of Mathematical and Statistical Psychology*, 63(3):509–525, 2010. [p183, 184, 188]

P. Lovie and A. D. Lovie. Charles Edward Spearman, F.R.S. (1863–1945). *Notes and Records of the Royal Society*, 50(1):75–88, 1996. doi: 10.1098/rsnr.1996.0007. [p198]

D. Magis. A note on the item information function of the four-parameter logistic model. *Applied Psychological Measurement*, 37(4):304–315, 2013. [p188]

M. C. Neale, M. D. Hunter, J. N. Pritikin, M. Zahery, T. R. Brick, R. Kirkpatrick, R. Estabrook, T. C. Bates, H. Maes, and S. M. Boker. **OpenMx** 2.0: Extended structural equation and statistical modeling. *Psychometrika*, in press. doi: 10.1007/s11336-014-9435-8. [p182]

B. Nosek, G. Alter, G. Banks, D. Borsboom, S. Bowman, S. Breckler, S. Buck, C. Chambers, G. Chin, G. Christensen, et al. Promoting an open research culture. *Science*, 348(6242):1422–1425, 2015. [p182]

D. Oakes. Direct calculation of the information matrix via the EM algorithm. *Journal of the Royal Statistical Society B (Statistical Methodology)*, 61(2):479–482, 1999. [p189]

M. Orlando and D. Thissen. Likelihood-based item-fit indices for dichotomous Item Response Theory models. *Applied Psychological Measurement*, 24(1):50–64, 2000. [p193]

R. D. Peng. Reproducible research in computational science. *Science*, 334(6060):1226, 2011. [p182]

J. N. Pritikin. *ifaTools: Toolkit for Item Factor Analysis with OpenMx*, 2015a. URL `https://CRAN.R-project.org/package=ifaTools`. R package version 0.8. [p186]

J. N. Pritikin. *rpf: Response Probability Functions*, 2015b. URL `https://CRAN.R-project.org/package=rpf`. R package version 0.51. [p191]

J. N. Pritikin, M. D. Hunter, and S. M. Boker. Modular open-source software for Item Factor Analysis. *Educational and Psychological Measurement*, 75(3):458–474, 2015. doi: 10.1177/0013164414554615. [p182]

G. Rasch. *Probabilistic Models for Some Intelligence and Attainment Tests*. MESA Press, 1960/1993. [p198]

RStudio and Inc. *shiny: Web Application Framework for R*, 2014. URL `http://CRAN.R-project.org/package=shiny`. R package version 0.10.2.1. [p182]

F. Samejima. Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph Supplement*, 34(4):100, 1969. [p183]

D. Thissen and L. Steinberg. Data analysis using item response theory. *Psychological Bulletin*, 104(3): 385–395, 1988. [p195]

D. Thissen, L. Steinberg, and J. A. Mooney. Trace lines for testlets: A use of multiple-categorical-response models. *Journal of Educational Measurement*, 26(3):247–260, 1989. [p187]

D. Thissen, L. Cai, and R. D. Bock. *The Nominal Categories Item Response Model*, pages 43–75. Routledge, 2010. [p183, 184, 189, 195]

W. M. Yen. Scaling performance assessments: Strategies for managing local item dependence. *Journal of Educational Measurement*, 30(3):187–213, 1993. [p192]

Joshua N. Pritikin
Department Psychology
University of Virginia
Charlottesville, VA 22904 USA
jpritikin@virginia.edu

Karen M. Schmidt
Department Psychology
University of Virginia
Charlottesville, VA 22904 USA
kschmidt@virginia.edu

Spatio-Temporal Interpolation using gstat

by Benedikt Gräler, Edzer Pebesma and Gerard Heuvelink

Abstract We present new spatio-temporal geostatistical modelling and interpolation capabilities of the R package **gstat**. Various spatio-temporal covariance models have been implemented, such as the separable, product-sum, metric and sum-metric models. In a real-world application we compare spatio-temporal interpolations using these models with a purely spatial kriging approach. The target variable of the application is the daily mean PM_{10} concentration measured at rural air quality monitoring stations across Germany in 2005. R code for variogram fitting and interpolation is presented in this paper to illustrate the workflow of spatio-temporal interpolation using **gstat**. We conclude that the system works properly and that the extension of **gstat** facilitates and eases spatio-temporal geostatistical modelling and prediction for R users.

Introduction

The collection and processing of spatio-temporal data is rapidly increasing due to technological advances and the societal need for analysis of variables that vary in space and time, such as weather and air quality variables, and crop yields. Analysis of spatial and temporal correlations is useful in itself to get insight into the character and causes of variability, but they are also important to predict values at points from neighbouring observations. Spatio-temporal interpolation can potentially provide more accurate predictions than spatial interpolation because observations taken at other times can be included. In addition, spatio-temporal interpolation allows predictions to be made at single locations or entire fields in between and beyond observation times. However, adding the temporal domain implies that variability in space and time must be modelled, which is more complicated than modelling purely spatial or purely temporal variability. The spatial, temporal and spatio-temporal dependence structures, for instance represented as variograms, do not necessarily coincide with each other in terms of their parameters nor in terms of their family. In the simplest case, a spatio-temporal anisotropy parameter might be enough to deal with the different dependence structures, but this poses strong assumptions on the process.

Interpolation of spatial random fields is a common task in geostatistics. Simple approaches like inverse distance weighted predictions or the well known kriging procedures have routinely been applied for many years. Nowadays, modern sensors allow to monitor different variables at an increasing temporal resolution producing rich spatio-temporal data sets. This calls as well for theory and methods to deal with these data sets to gain a better understanding of the observed spatio-temporal processes. While the theoretical aspects of spatio-temporal geostatistics show good progress (Cressie and Wikle, 2011), implementations lack behind. This hinders a wide application of spatio-temporal modelling, as typically extensive scripting and thorough understanding is necessary to build spatio-temporal models. Handling of spatio-temporal data in R is provided by the **spacetime** package (Pebesma, 2012). In this paper, we present an extension of the **gstat** package (Pebesma, 2004) (version 1.1-3) that reuses the **spacetime** classes for the estimation of spatio-temporal covariance/variogram models and to perform spatio-temporal interpolation. Our implementation handles various types of spatio-temporal covariance structures and facilitates spatio-temporal interpolation. The notation of functions in **gstat** is extended in a way closely following the purely spatial design. This allows a researcher acquainted with **gstat** to readily use spatio-temporal tools. The use of the newly implemented functions is presented and illustrated by mapping spatio-temporal air-quality data. Another package that offers extensive spatio-temporal geostatistical functionality is **RandomFields** (Schlather et al., 2014); further packages are mentioned in the CRAN Task View on Handling and Analyzing Spatio-Temporal Data[1].

The paper is organised as follows. The next section introduces the general interpolation routine and describes the different spatio-temporal covariance models, followed by a section introducing the German rural background data set for 2005 and performing the parameter estimation (i.e. covariance model fitting). Cross-validation results are presented and discussed in the section thereafter. Conclusions are drawn in the closing section. R scripts reproducing this study are available from within the **gstat** package as demos. 'stkrige' re-estimates the variogram models, 'stkrige-prediction' re-executes the prediction for a time series and a couple of stations, and 'stkrige-crossvalidation' re-runs the entire leave-one-out cross-validation (note that the latter takes a few hours).

[1] https://CRAN.R-project.org/view=SpatioTemporal

Spatio-temporal dependence modelling and kriging

In the following, we will assume a Gaussian spatio-temporal random field Z defined over a spatial domain $\mathcal{S}$ and temporal domain $\mathcal{T}$. Typically, a sample $\mathbf{z} = (z(s_1, t_1), \ldots, z(s_n, t_n))$ has been observed at a set of distinct spatio-temporal locations $(s_1, t_1), \ldots, (s_n, t_n) \in \mathcal{S} \times \mathcal{T} \subseteq \mathbb{R}^2 \times \mathbb{R}$ that may include repeated measurements at the same location or simultaneous measurements at multiple spatial locations. Often, one is interested in modelling Z from the sample $\mathbf{z}$ in order to predict at unobserved locations in space and time or simulate from the conditional distribution.

Across our domain of interest $\mathcal{S} \times \mathcal{T}$, we assume the random field Z to be stationary and spatially isotropic. Hence, the field can be characterised through a mean μ and a covariance function C_{st} where the spatio-temporal covariance only depends on the separating distances across space $h \in \mathbb{R}_{\geq 0}$ and time $u \in \mathbb{R}_{\geq 0}$. Note that extensions beyond this set-up can easily be derived as has been done for the pure spatial case using for instance universal kriging to overcome the stationarity of the mean. The general spatio-temporal covariance function is given by $C_{\text{st}}(h, u) = \text{Cov}\left(Z(s, t), Z(\tilde{s}, \tilde{t})\right)$ for a separating spatial distance h and temporal distance u and any pair of points $(s, t), (\tilde{s}, \tilde{t}) \in \mathcal{S} \times \mathcal{T}$ with $\|s - \tilde{s}\| = h$ and $|t - \tilde{t}| = u$. In general, this covariance function is hard to estimate but a couple of models using simplifying assumptions will be presented in the following together with their spatio-temporal variograms $\gamma_{\text{st}}(h, u) = C_{\text{st}}(0, 0) - C_{\text{st}}(h, u)$ and encoding in **gstat**. Given a valid covariance function, the covariance matrices used in the linear predictor are easily obtained and the same algebraic operations as in the well known spatial case yield predictions of the desired random field (Cressie and Wikle, 2011). A major difference is, however, the computational complexity of the matrix inversion. Typically, observations are made at a rather high temporal frequency leading to a number of spatio-temporal locations that is too large for global kriging. Hence, interpolation based on a selected neighbourhood of a subset of all data points becomes beneficial. Additionally, this relaxes the assumption of stationarity, as smooth variations in the mean value across the domain can be respected. The related class of dynamic models also addresses the computational complexity resulting in a temporal Markov structure. Implementations can be found in **spTimer** by Bakar and Sahu (2015), **spBayes** by Finley et al. (2015), **spate** by Sigrist et al. (2015) or **INLA** by Lindgren and Rue (2015).

Covariance models

The covariance models implemented in **gstat** and presented in this paper are introduced in the following. Besides further extensions we focus on the basic classes of the *separable*, *product-sum*, *metric* and *sum-metric* spatio-temporal covariance functions. The building blocks (in the following denoted as `spatialVgm`, `temporalVgm` or `jointVgm`) of the spatio-temporal covariance functions are any of the purely spatial variogram models already available in **gstat**. Each one of the building blocks is created by a call of the function `gstat::vgm()`. Remaining arguments such as `sill` (the joint sill), `nug` (the joint nugget component) or `stAni` (the spatio-temporal anisotropy used in `jointVgm`) are scalars and refer to parameters of the entire spatio-temporal covariance function:

a) The *separable covariance model* assumes that the spatio-temporal covariance function can be represented as the product of a spatial and temporal term:

$$C_{\text{sep}}(h, u) = C_{\text{s}}(h) C_t(u)$$

Its variogram is given by (see Appendix for details):

$$\gamma_{\text{sep}}(h, u) = \text{sill} \cdot (\bar{\gamma}_s(h) + \bar{\gamma}_t(u) - \bar{\gamma}_s(h)\bar{\gamma}_t(u))$$

where $\bar{\gamma}_s$ and $\bar{\gamma}_t$ are standardised spatial and temporal variograms with separate nugget effects and (joint) sill of 1. The overall sill parameter is denoted by "sill".

The R package **gstat** encodes this model as:

```
vgmST("separable", space = spatialVgm, time = temporalVgm, sill = sill)
```

The separable model has a strong computational advantage in the setting where each spatial location has an observation at each temporal instance (a "STFDF" object without 'NA's, Pebesma 2012). In these cases, the covariance matrix (and its inverse) can be composed using the Kronecker-product of the purely spatial and purely temporal covariance matrices (respectively their inverse).

b) The above model extends to the *product-sum covariance model* that we give here in a slightly different notation as De Cesare et al. (2001) and De Iaco et al. (2001) by

$$C_{\text{ps}}(h, u) = k C_{\text{s}}(h) C_{\text{t}}(u) + C_{\text{s}}(h) + C_{\text{t}}(u)$$

with $k > 0$.

The corresponding variogram can be written as

$$\gamma_{ps}(h,u) = (k \cdot \text{sill}_t + 1)\,\gamma_s(h) + (k \cdot \text{sill}_s + 1)\,\gamma_t(u) - k\gamma_s(h)\gamma_t(u)$$

where γ_s and γ_t are spatial and temporal variograms (see Appendix for details). The parameter k needs to be positive and the following identity defines the overall sill (sill_{st}) of the model in terms of the model's spatial and temporal sills:

$$\text{sill}_{st} = k \cdot \text{sill}_s \cdot \text{sill}_t + \text{sill}_s + \text{sill}_t$$

The above equation can also be used to estimate k based on the three sill values. An alternative formulation of the product-sum variogram can be found in De Iaco et al. (2001).

The **gstat** definition of this model reads:

```
vgmST("productSum", space = spatialVgm, time = temporalVgm, k = k)
```

c) Assuming identical spatial and temporal covariance functions except for spatio-temporal anisotropy, allows to use a spatio-temporal *metric covariance model* where, after matching space and time by an anisotropy correction κ (`stAni`), the spatial, temporal and spatio-temporal distances are treated equally resulting in a single covariance model C_{joint}:

$$C_m(h,u) = C_{joint}\left(\sqrt{h^2 + (\kappa \cdot u)^2}\right)$$

The variogram evaluates to

$$\gamma_m(h,u) = \gamma_{joint}\left(\sqrt{h^2 + (\kappa \cdot u)^2}\right)$$

where γ_{joint} (`jointVgm`) is any known variogram that may include a nugget effect. The following line generates the model in **gstat**:

```
vgmST("metric", joint = jointVgm, stAni = stAni)
```

The spatio-temporal anisotropy parameter κ (`stAni`) is given as spatial unit per temporal unit. In many cases, this will be in m/second, as these are the base units in our implementation. All temporal distances are hence internally re-scaled to an equivalent spatial distance using `stAni` and treated as metric 3D-space.

d) A combination of spatial, temporal and a metric model including an anisotropy parameter κ is found in Bilonick (1988) and revisited by Snepvangers et al. (2003) as the *sum-metric covariance model*:

$$C_{sm}(h,u) = C_s(h) + C_t(u) + C_{joint}\left(\sqrt{h^2 + (\kappa \cdot u)^2}\right)$$

This model allows for spatial, temporal and joint nugget effects. Thus, the variogram is given by

$$\gamma_{sm}(h,u) = \gamma_s(h) + \gamma_t(u) + \gamma_{joint}\left(\sqrt{h^2 + (\kappa \cdot u)^2}\right)$$

where γ_s, γ_t and γ_{joint} are spatial, temporal and joint variograms with separate nugget-effects. This model can be defined in **gstat** through:

```
vgmST("sumMetric", space = spatialVgm, time = temporalVgm, joint = jointVgm,
      stAni = stAni)
```

e) A simplified version of the above model is to restrict the spatial, temporal and joint variograms to nugget free models. Additionally, a single spatio-temporal nugget is introduced and the variogram takes the representation:

$$\gamma_{ssm}(h,u) = \text{nug} \cdot \mathbf{1}_{h>0 \vee u>0} + \gamma_s(h) + \gamma_t(u) + \gamma_{joint}\left(\sqrt{h^2 + (\kappa \cdot u)^2}\right)$$

The *simple sum-metric covariance model* can be obtained by:

```
vgmST("simpleSumMetric", space = spatialVgm, time = temporalVgm,
      joint = jointVgm, nugget = nug, stAni = stAni)
```

Note that the above mentioned spatial, temporal and joint components of the spatio-temporal covariance and variogram models need not necessarily exhibit the same structure. Taking for instance the product-sum and sum-metric models that both contain single temporal and spatial variogram models: the best fits of the respective spatio-temporal models might suggest different variogram families and

parameters for the pure spatial and temporal ones. This is due to the target of finding the best overall variogram surface resulting in (potentially) different marginal models.

Parameter estimation

Fitting routines for the above variogram models are implemented in **gstat** through the function `fit.StVariogram()`, which calls `optim()` from the R core package **stats**. Additional parameters to improve the numerical optimisation can be provided to `fit.StVariogram()` and will be passed on to `optim()` (using R's three-dots mechanism). As some of the parameters are limited to certain ranges (nuggets need to be non-negative, ranges must be positive), it is advised to use an optimisation routine that allows to impose limits on the search space (i.e. `"L-BFGS-B"`) and provide sensible limits via `lower` and `upper`. By default, the method `"L-BFGS-B"` is called and the smallest lower and largest upper bounds supported by the model are given. The estimation of the spatio-temporal variogram models relies on a sample variogram empirically derived from the data. In contrast to the spatial variogram line, the spatio-temporal variogram is represented by a surface for lag-classes composed of a spatial and temporal separation distance. Different from the spatial case, a spatio-temporal sample variogram contains lag-classes of zero spatial separation describing pure temporal dependencies. Without duplicate observations, no estimates can be made for the lag-class with both zero spatial and zero temporal separation. The sample variogram is calculated through the function `variogram()` that dispatches the call for spatio-temporal data objects (of class "STFDF", "STSDF", or "STIDF") from **spacetime**. For a visual judgement of the fit between sample and fitted variograms the `plot()` function can be called to show the variogram surfaces next to each other as coloured level plots. Alternatively, a wireframe plot is obtained by setting the parameter `wireframe = TRUE` (Figure 3). A further option is to plot the differences between the sample and model variogram surfaces by setting `diff = TRUE`, see Figure 4. Additionally to visual comparison, `fit.StVariogram()` provides the output of `optim` as attribute `optim.out` of the returned S3 class "StVariogram". This attribute includes valuable information to diagnose the success of the `optim` routine. It contains for instance the convergence code (`$convergence`) or message (`$message`) and the optimised value (`$value`), which is the mean of the (weighted) squared deviations between sample and fitted variogram surfaces. Furthermore, it is advised to check the estimated parameters against the parameter boundaries and starting values. Additionally, starting values might also influence the success and result of the optimisation, as local optima may occur due to the interdependence of the parameters. Alternatively, the user might want to start a grid search in order to better asses the sensitivity of the estimates.

The fitting approach is identical for all covariance models. However, with the flexibility of the model also the number of parameters increases, making a numerical estimation at times cumbersome. Starting values can in most cases be read from the sample variogram. Parameters of the spatial and temporal variograms can be assessed from the spatio-temporal surface fixing the counterpart at 0. The overall spatio-temporal sill including the nugget can be deducted from the plateau that a nicely behaving sample variogram reaches for "large" spatial and temporal distances. An important issue is the potentially different orders of magnitude of the parameters. It is at times advisable to rescale spatial and temporal distances to ranges similar to the ones of sills and nuggets using the parameter `parscale`. `parscale` needs to be provided via `control = list(parscale = . . . )` and holds a vector of the same length as the number of parameters to be optimised (see the documentation of `optim` for further details).

Currently, the implemented fitting routines are based on the (weighted) mean squared difference between model and sample variogram surfaces. By default, all values are associated the same weight (`fit.method = 6`), but other options are available that allow for different weighting schemes based on the number of pairs, spatial, temporal and spatio-temporal distances or the variogram's value. Table 1 lists all currently implemented options. Depending on the target neighbourhood size of the desired interpolation, it might be beneficial to narrow down the spatial and temporal distances and to introduce a cutoff. This ensures that the model is fitted to the differences over space and time actually used in the interpolation, and reduces the risk of overfitting the variogram model to large distances not used for prediction. Please note that methods 2 and 10 (Table 1) involve weights based on the fitted variogram that might lead to bad convergence properties of the parameter estimates. Furthermore, the scalar `stAni` in methods 7 and 11 will either be the actual fitted spatio-temporal anisotropy if it is included in the model or a fixed value that has to be passed as `stAni` by the user to `fit.StVariogram`. The latter is advised, as the former might lead to bad convergence properties as in the case of weights based on the fitted variogram mentioned above. As the estimation of an anisotropy scaling might be cumbersome on a visual basis, we provide the function `estiStAni` that provides estimates based on the empirical spatio-temporal variogram. Four heuristics are available based on (i) rescaling a linear model (`linear`), (ii) estimating equal ranges (`range`), (iii) rescaling a pure spatial variogram (`vgm`) or (iv) estimating a complete spatio-temporal metric variogram model and returning its spatio-temporal anisotropy parameter (`metric`). The choice of the weighting scheme will influence the selected model

`fit.method`	weights
0	no fitting
1 and 3	N_j
2 and 4	$N_j/\gamma\left(h_j, u_j\right)^2$
5	reserved for REML
6	1, no weighting
7	$N_j/\left(h_j^2 + \text{stAni}^2 \cdot u_j^2\right)$
8	N_j/h_j^2
9	N_j/u_j^2
10	$1/\gamma\left(h_j, u_j\right)^2$
11	$1/\left(h_j^2 + \text{stAni}^2 \cdot u_j^2\right)$
12	$1/h_j^2$
13	$1/u_j^2$

Table 1: List of implemented weighting schemes for variogram optimisation. Methods 3, 4, and 5 are kept for compatibility reasons with the purely spatial `fit.variogram` function. The following notation is used: N_j number of pairs, h_j mean spatial distance and u_j mean temporal distance for each bin j, γ the actual proposed variogram model and `stAni` a spatio-temporal anisotropy scaling.

and different weightings might be further assessed in a cross-validation of the selected model. To increase numerical stability, it is advisable to use weights that do not change with the current model fit.

Kriging

Standard kriging (`krigeST`) and trans Gaussian kriging (`krigeSTTg`) have been implemented. As spatio-temporal kriging based on the complete data set might be too computationally expensive, local kriging is an attractive alternative. This poses the question of how to select the "nearest" neighbours from the spatio-temporal space $\mathcal{S} \times \mathcal{T}$. A natural choice would be to select the spatio-temporal locations exhibiting the strongest correlation to the unobserved location. Depending on the spatio-temporal covariance model, the relation between spatial and temporal distance in determining the strength of correlation will vary. As a proxy, we use a spatio-temporal anisotropy parameter that relates spatial and temporal distances in the same manner as in the metric covariance models. The k-nearest neighbours within this metric spatio-temporal space $\mathcal{S} \times \mathcal{T}$ are selected using the R package **FNN** (Beygelzimer et al., 2013). The interpolation performs iteratively for each spatio-temporal prediction location with a local subset of the data set. Without neighbourhood selection, kriging uses all data.

As the metric induced by the spatial and rescaled temporal distances are only proxies to the strength of correlation between locations (see Figure 1), we provide an option to search a larger metric neighbourhood. Within this larger neighbourhood, the covariance function is evaluated for all spatio-temporal locations and the neighbouring locations with the largest covariance values are then selected for prediction. However, this approach might still suffer from a clustering of locations and alternatives such as a staged search (find spatial neighbours first and select a set of temporal instances for each spatial neighbour) or an octant search (select neighbours per spatial quadrant from preceding and following time stamps separately) could be considered. However, these alternatives are not yet available in **gstat**.

Application and illustration

The data set used is taken from AirBase[2], the air quality data base for Europe provided by the European Environmental Agency (EEA). We focus on a single air quality indicator, particulate matter with a diameter less than 10 μm, measured at rural background stations for 2005 (PM_{10}). The data base contains data for many years. Besides rural, also urban areas are monitored and not only at background locations (e.g. traffic stations). However, these processes are considered to be of a different nature and should be treated separately. As a use case, we therefore limit our data set to the rural background stations in Germany. Figure 2 shows for 8 randomly chosen days daily mean

[2] AirBase – The European air quality database

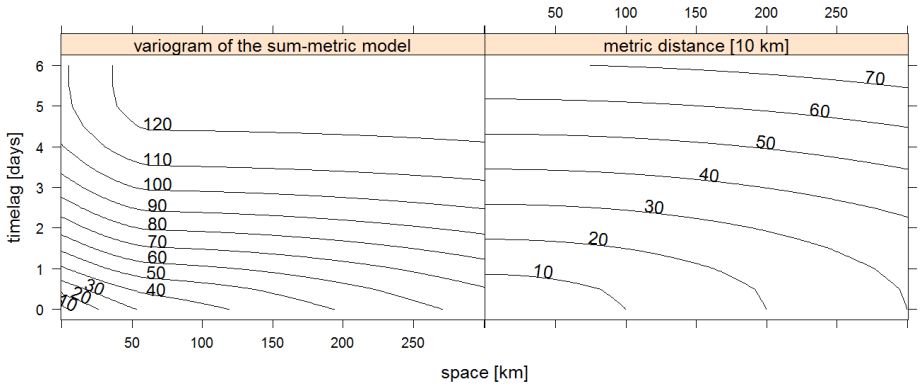


Figure 1: A contourplot showing how the spatio-temporal sum-metric variogram model (as estimated in the application below) and a metric distance relate to each other. Distances are rescaled by 1/5 for easy plotting.

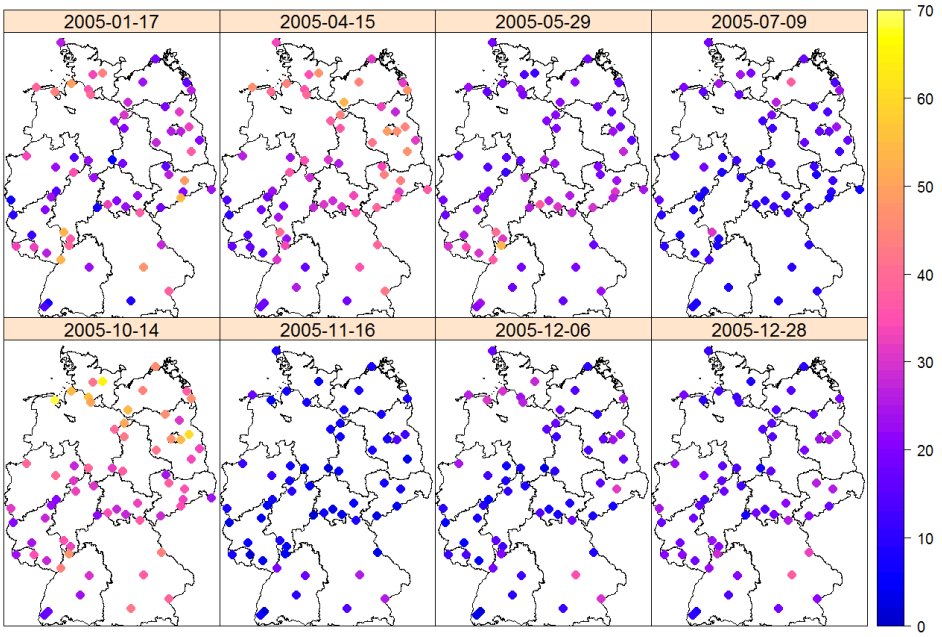


Figure 2: Daily mean PM_{10} concentration $[\mu g/m^3]$ at 8 randomly selected days in 2005.

values of PM_{10} concentrations for the entire monitoring network over Germany in 2005 with 69 rural background stations.

In order to fit a spatio-temporal model to the air quality data set, the empirical variogram surface is computed and used as input for the fitting routines of the different models. The empirical variogram is based on spatio-temporal bins that span regularly over space and time.

Regular measurements over time (i.e. hourly, daily) motivate regular binning intervals of the same temporal resolution. Nevertheless, flexible binning boundaries can be passed for spatial and temporal dimensions. This allows for instance to use smaller bins at small distances and larger ones for large distances. Temporal boundaries, instead of lags, are required when the sampling of the data is non-regular. In cases where regular temporal observations can be assumed, this is utilised in the sample variogram calculations and any two temporal consecutive observations are assumed to have the same temporal distance. Figure 3 shows the empirical variogram along with the proposed best fitting model of each spatio-temporal variogram family as perspective wireframe plots. In order to better identify structural shortcomings of the selected model, a difference plot (Figure 4) is a helpful visual diagnostic plot.

Beyond the selection of the spatio-temporal variogram family, each component of this model can be chosen from any implemented one-dimensional variogram. In Table 2 a selection of fitted

models in terms of their residuals compared to the sample variogram surface is shown. The best fitting spatio-temporal model of each family is given as:

a) separable model (weighted MSE: 6.82):

	partial sill	model	range	nugget	sp.-temp. sill
space	0.86	Exp	558 km	0.14	124
time	1.00	Sph	5.6 days	0.00	

obtained via:

```
separableModel <- vgmST("separable", space = vgm(0.9, "Exp", 200, 0.1),
                        time = vgm(0.9, "Sph", 3.5, 0.1), sill = 124)
fit.StVariogram(empVgm, separableModel, fit.method = 7, stAni = 117.3,
                method = "L-BFGS-B",
                control = list(parscale = c(100, 1, 10, 1, 100)),
                lower = c(10, 0, 0.1, 0, 0.1), upper = c(2000, 1, 12, 1, 200))
```

b) product-sum model (weighted MSE: 6.91)

	partial sill	model	range	nugget	k
space	6.8	Exp	542 km	1.2	1.61
time	8.7	Sph	5.5 days	0.0	

obtained via

```
prodSumModel <- vgmST("productSum", space = vgm(10, "Exp", 200, 1),
                      time = vgm(10, "Sph", 2, 1), k = 2)
fit.StVariogram(empVgm, prodSumModel, fit.method = 7, stAni = 117.3,
                method = "L-BFGS-B", lower = rep(0.0001, 7)
                control = list(parscale = c(1, 10, 1, 1, 0.1, 1, 10)))
```

c) metric model (weighted MSE: 10.05)

	partial sill	model	range	nugget	anisotropy
joint	123.4	$\text{Mat}_{\kappa=0.6}$	453 km	17.4	189 km/day

obtained via

```
metricModel <- vgmST("metric", joint = vgm(60, "Mat", 150, 10, kappa = 0.6),
                     stAni = 60)
fit.StVariogram(empVgm, metricModel, fit.method = 7, stAni = 117.3,
                method = "L-BFGS-B", control = list(parscale = c(10, 20, 5, 10)),
                lower = c(80, 50, 5, 50), upper = c(200, 1500, 60, 300))
```

d) sum-metric model (weighted MSE: 3.31)

	partial sill	model	range	nugget	anisotropy
space	16.4	Sph	67 km	0	
time	9.3	Exp	0.9 days	0	
joint	91.5	Sph	999 km	7.3	185 km/day

obtained via

```
sumMetricModel <- vgmST("sumMetric", space = vgm(20, "Sph", 150, 1),
                        time = vgm(10, "Exp", 2, 0.5),
                        joint = vgm(80, "Sph", 1500, 2.5), stAni = 120)
fit.StVariogram(empVgm, sumMetricModel, fit.method = 7, stAni = 117.3,
                method = "L-BFGS-B",
                control = list(parscale = c(1, 100, 1, 1, 0.5, 1, 1, 100,
                                            1, 100),
                               maxit = 10000),
                lower = c(sill.s = 0,  range.s = 10, nugget.s = 0,
                          sill.t = 0, range.t = 0.1, nugget.t = 0,
                          sill.st = 0, range.st = 10, nugget.st = 0, anis = 40),
                upper = c(sill.s = 200, range.s = 1000, nugget.s = 20,
                          sill.t = 200, range.t = 75, nugget.t = 20,
                          sill.st = 200, range.st = 5000, nugget.st = 20,
                          anis = 500))
```

model	joint	Exp+Exp	Exp+Sph	Sph+Exp	Sph+Sph	$\text{Mat}_{\kappa=0.6}$
separable	·	9.87	**6.82**	10.42	7.50	·
product-sum	·	10.09	**6.91**	10.64	7.59	·
metric	·	10.25	·	·	10.59	**10.05**
sum-metric	Exp	4.10	3.60	3.89	3.32	·
	Sph	3.74	3.73	**3.31**	3.36	·
simple sum-metric	Exp	4.10	3.60	3.94	3.32	·
	Sph	3.74	3.98	**3.31**	3.56	·

Table 2: Weighted MSE (`fit.method = 7`, see Table 1) for different spatio-temporal variogram families and different choices for the one-dimensional variogram components. Columns denote the spatial and temporal variogram choices. The metric model is only applicable if both domains use the same family.

e) simple sum-metric model (weighted MSE: 3.31)

	partial sill	model	range	anisotropy	sp.-temp. nugget
space	16.4	Sph	67 km		} 7.3
time	9.3	Exp	0.9 days		
joint	91.5	Sph	999 km	185 km/day	

obtained via

```
simpleSumMetricModel <- vgmST("simpleSumMetric", space = vgm(120, "Sph", 150),
                              time = vgm(120, "Exp", 10),
                              joint = vgm(120, "Sph", 150),
                              nugget = 10, stAni = 150)
fit.StVariogram(empVgm, simpleSumMetricModel, fit.method = 7,
                stAni = 117.3, method = "L-BFGS-B",
                control = list(parscale = c(1, 10, 1, 1, 1, 100, 1, 10))
                lower = c(sill.s = 0, range.s = 10,
                          sill.t = 0, range.t = 0.1,
                          sill.st = 0, range.st = 10,
                          nugget = 0, anis = 40),
                upper = c(sill.s = 200,  range.s = 500,
                          sill.t = 200,  range.t = 20,
                          sill.st = 200, range.st = 5000#,
                          nugget = 100, anis = 1000))
```

The variogram parameters are numerically optimised using the function `fit.StVariogram` and the `"L-BFGS-B"` routine of `optim`. The parameter `fit.method` that controls the weighing of the residuals between empirical and model surface of `fit.StVariogram` is set to 7 (the spatio-temporal analog to the commonly used spatial weighting). A full list of all weighting schemes is presented in Table 1. In our application, the residuals are multiplied by the number of pairs in the corresponding spatio-temporal bin divided by the metric distance: $N_j/(h_j^2 + \text{stAni}^2 \cdot u_j^2)$. The spatio-temporal anisotropy is estimated beforehand and fixed at 118 km/day. This weighting scheme puts higher confidence in lags filled with many pairs of spatio-temporal locations, but respects to some degree the need of an accurate model for short distances, as these short distances are the main source of information in the prediction step. Note, that different weighting schemes will in general result in different model parameters generating different interpolation values. Our selection is based on the assumption that well filled bins provide more reliable empirical variogram estimates and the fact that short distances are the most important ones for a local interpolation.

For comparison with classical approaches, we interpolate across Germany iteratively for each single day using all available data for variogram estimation. The purely spatial empirical variogram can directly be obtained from the empirical spatio-temporal variogram, by fixing the temporal lag at 0 separation. From the same set of variogram models as investigated for the spatio-temporal models, the exponential model (partial sill: 66.5, range: 224 km, nugget: 13.5) is the best suited based on the optimisation criterion. Alternatively, we could have fitted the spatial variogram for each day separately using observations from that day only. However, given the small number of observation stations, this produced unstable variograms for several days and we decided to use the single spatial variogram derived from all spatio-temporal locations treating time slices as uncorrelated copies of the spatial random field.

Once the best fitting spatio-temporal variogram model is identified, the interpolation can be

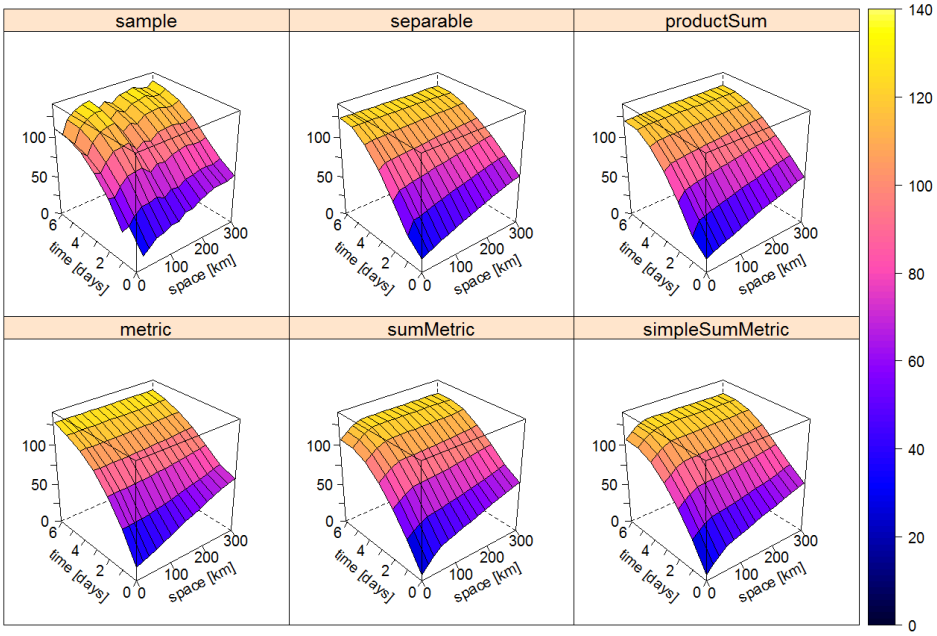


Figure 3: Sample and the best fitting spatio-temporal variogram of each family.

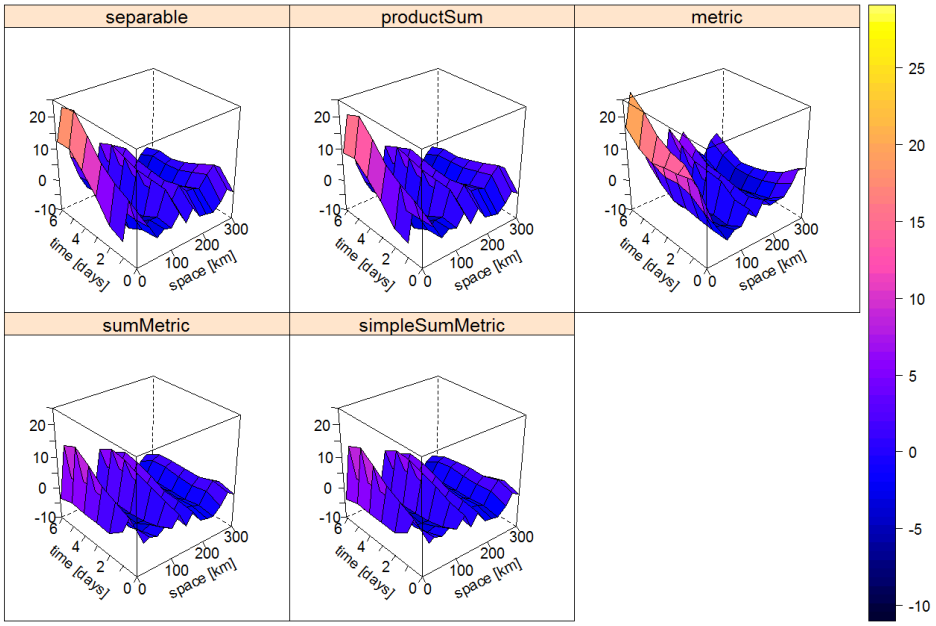


Figure 4: Differences between the sample and the best fitting spatio-temporal variogram of each family.

executed with the help of the function `krigeST`. We use the sum-metric model that obtained the smallest RMSE (compare Table 2) to produce a gridded prediction. The interpolation domain consists of daily values for a regular grid spanning over Germany in UTM projection. The cell size is 10 km $\times$ 10 km. Figure 5 shows the interpolated grid for the same days as Figure 2 alongside with all sampling locations. Additionally, maps depicting the differences from a leave-one-out cross-validation are presented in Figure 6. A time series view is presented in Figure 7 showing the observed and predicted time series at a single location along with its 95 % prediction intervals. An animation of the entire year of daily mean PM_{10} prediction maps can be viewed online.[3].

The interpolated maps are generated for a set of time stamps `tIDs` and a grid over Germany

[3] http://gstat.r-forge.r-project.org/STpred.html

covariance model	wMSE	neigh.	RMSE	MAE	ME	COR
pure Spatial		10	6.15	4.09	−0.01	0.84
separable	[6.82]	10	6.08	4.04	−0.01	0.84
product-sum	[6.91]	10	6.08	4.04	−0.01	0.84
metric	[10.05]	10	6.11	4.07	0.03	0.84
sum-metric	[3.31]	10	6.16	4.08	−0.06	0.84
simple sum-metric	[3.31]	10	6.14	4.08	−0.02	0.84
pure Spatial		50	6.10	4.07	0.00	0.84
separable	[6.82]	50	6.05	4.04	0.01	0.84
product-sum	[6.91]	50	6.05	4.04	0.00	0.84
metric	[10.05]	50	6.07	4.08	0.03	0.84
sum-metric	[3.31]	50	6.14	4.09	−0.01	0.84
simple sum-metric	[3.31]	50	6.14	4.08	−0.02	0.84

Table 3: Leave-one-out cross-validation results. The column wMSE refers to the optimised value from the variogram estimation.

`DE_pred` by

```
krigeST(PM10 ~ 1, data = DE_RB_2005[, tIDS], newdata = DE_pred,
        fitSumMetricModel, nmax = 50, stAni = fitMetricModel$stAni / 24 / 3600)
```

To further compare the different approaches, a leave-one-out cross-validation was carried out. The spatio-temporal interpolations are done for the closest 50 and 10 neighbours assessing the impact of the neighbourhood size. Inspection of the ranges of the variograms in the temporal domain, suggests that any station more than at least 6 days apart does not meaningfully contribute. Furthermore, the local estimation allows the spatio-temporal random field to have a varying mean value over space and time. The purely spatial interpolation can be considered as the extreme temporally local case, where only observations from the same time instance are considered.

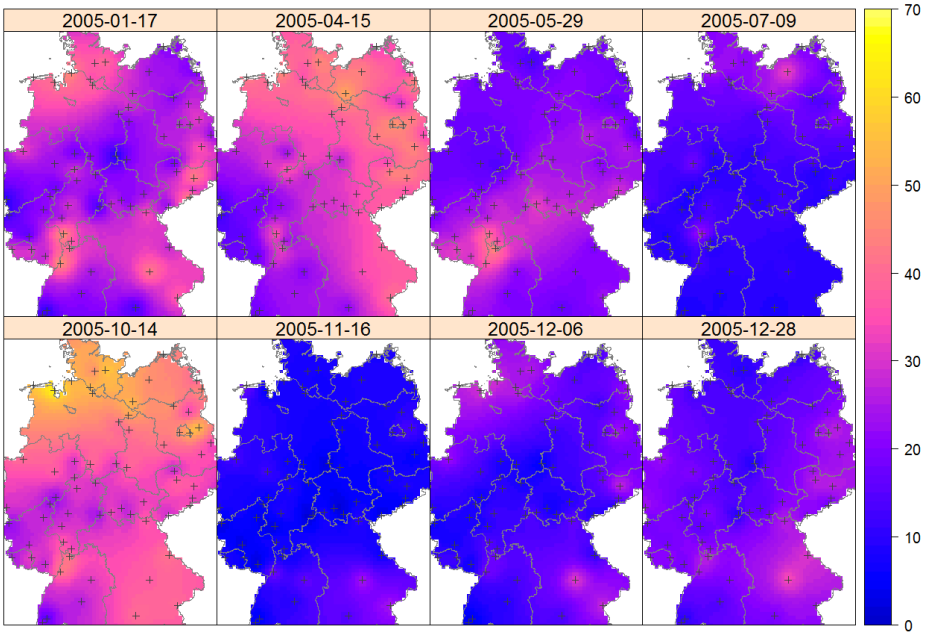


Figure 5: Spatio-temporal interpolation of daily mean PM_{10} concentrations using the sum-metric covariance model with the closest 50 neighbouring spatio-temporal locations. The crosses indicate sampling locations. The cell size of the grid in UTM projection is 10 km × 10 km.

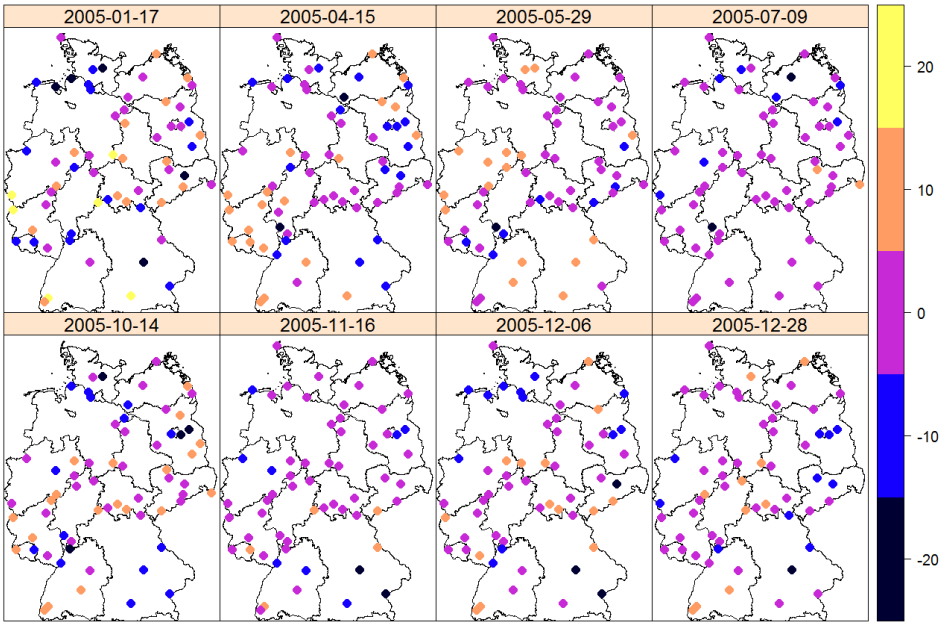


Figure 6: Differences of spatio-temporal predictions and observed daily mean PM_{10} concentrations using the sum-metric covariance model with the closest (approx. strongest correlated) 50 neighbouring spatio-temporal locations.

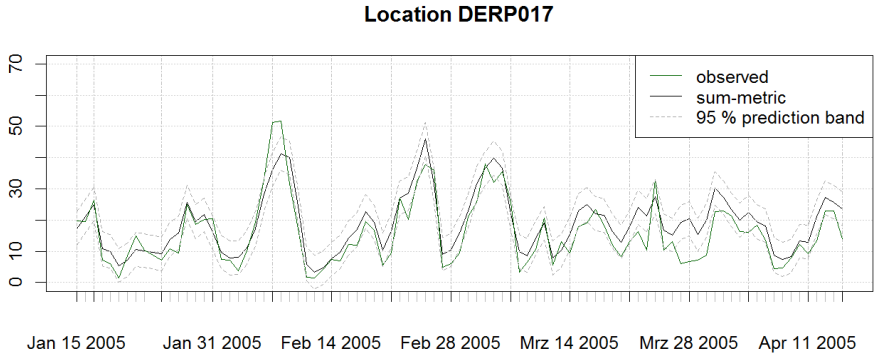


Figure 7: Subset of the time series of observed and predicted PM_{10} at a single station in Lower Saxony along with its 95 % prediction intervals.

Results and discussion

In terms of added value of spatio-temporal kriging measured in cross-validation results, Table 3 shows hardly any benefit in the illustrative example. This effect can to a large degree already be explained from the spatio-temporal variograms: a temporal lag of one or a few days leads already to a large variability compared to spatial distances of few hundred kilometres, implying that the temporal correlation is too weak to considerably improve the overall prediction. Nevertheless, investigating a process with a higher temporal frequency will likely show a stronger correlation in the temporal domain. Looking into station-wise cross-validation statistics (not shown), the four stations with an RMSE of 10 and larger correspond to the locations with the largest annual mean concentrations ($> 22\ \mu g/m^3$).

The added value of spatio-temporal kriging lies in the flexibility of the model. We are now in the position to not only interpolate at unobserved locations in space, but also at unobserved time instances. This makes spatio-temporal kriging a suitable tool to fill gaps in time series not only based on the time series solely, but also including some of its spatial neighbours. A very irregular sampled data set would as well largely benefit from a spatio-temporal approach, as spatially close but unobserved

locations in one time slice are not of any help in a purely spatial approach, but the spatio-temporal model would benefit from the observed value nearby at another time instance. In a completely regular data set, the distance to a spatio-temporal neighbour is at least as large as the pure spatial distance and hence the correlation is weaker. Furthermore, being able to capture the covariance structure over space and time might foster a better understanding of the process under study.

While we see spatio-temporal modelling as being a powerful tool, the cross-validation results in Table 3 show that spatio-temporal kriging will not solve the problems of all poorly spatially captured phenomena. Further preprocessing steps might be necessary to improve the modelling of this PM_{10} data set such as for instance a temporal AR-model followed by spatio-temporal residual kriging or using further covariates in a preceding (linear) modelling step. Providing the best possible model of PM_{10} concentrations across Germany was beyond the scope of this paper.

The selection of a spatio-temporal covariance model should not only be made based on the (weighted) mean squared difference between the empirical and model variogram surfaces (presented in Table 2), but also on conceptional choices and visual (Figure 3) judgement of the fits. Even though the function `fit.StVariogram` provides optimisation routines to fit different spatio-temporal variogram models, the numerical routines in the background may struggle to find the optimal parameters. Besides the lower and upper boundaries of the parameter space, the control parameter `parscale` of the `optim` function is a valuable option to improve the convergence of the optimisation. With passing `parscale` as entry of the list `control` a vector of scalars must be passed that controls the different levels of magnitude of the variogram parameters. In most applications, a change of 1 in the sills will have a stronger influence on the variogram surface than a change of 1 in the ranges. The problem becomes more difficult with an increasing number of parameters. In our application, using the simple sum-metric model as starting values for the full sum-metric model improved the convergence speed of the more complex model. In the presented application, the sum-metric model turns out to be the same as the simple sum-metric model. While this might at first sight be due to using the simpler model to generate starting values, different non simplified starting models converged to the same result.

Generally, it is important to keep in mind the strong interaction of the model parameters. It is typically not easy to distinguish how much of the spatio-temporal nugget and sill is attributed to spatial, temporal or joint components. In this paper we considered a joint numerical approach, but step-wise approaches where the components are estimated separately could as well be considered. The interested reader is also referred to Nash (2014). However, all optimisation approaches follow the premise that the studied process can be approximated with the given model and available data. If this premise fails, no optimal model can be selected.

An extension towards a restricted maximum likelihood method (REML) to fit the spatio-temporal variogram model would be desirable, as it overcomes some of the above mentioned drawbacks of the method of moments based approaches and would additionally provide standard errors for the parameter estimates. A REML approach would allow to take into account that sample variogram values are correlated. However, for large data sets (as in the spatio-temporal case), it is computationally more feasible to use a least squares fitting. To reduce the correlation of the variogram values, some randomisation could be implemented in large data sets, to calculate the sample variogram based on partially overlapping or even disjoint sets of observations.

The selected anisotropy as proxy to the relation of spatial and temporal distance in determining the strongest correlated neighbours might show a distortion for some models when only few neighbours are used towards the true set of the most correlated locations. However, this effect vanishes as soon as the spatio-temporal range of the model is sufficiently represented through the set of nearest neighbours.

As mentioned by Kyriakidis and Journel (1999), an alternative to space-time kriging might be co-kriging. However, this is only feasible if the number of time replicates is (very) small, as the number of cross-variograms to be modelled equals the number of *pairs* of time replicates. Also, co-kriging can only interpolate for these time slices, and not inbetween or beyond them. It does however provide prediction error covariances, which can help assessing the significance of estimated *change* parameters (Pebesma et al., 2005; Pebesma and Duin, 2005). Several of the space-time variograms presented here may be approximated by sets of direct variograms and cross-variograms.

Fitting variogram models to sample space-time variograms is in our implementation done by `stats::optim`. Our example script uses method `"L-BFGS-B"` and provides upper and lower parameter boundaries, e.g. to make sure sill parameters do not become negative. There has been a lot of research in optimisation since the development of the methods included in `optim`, some of which has been reported in the special issue of the Journal of Statistical Software (Varadhan, 2014), and we do see potential to improve the options in this respect.

The approximate selection of the most correlated neighbours solves the lack of a natural notion of a joint distance across space and time. However, other sampling properties might introduce a bias in the prediction. The prediction at an unobserved location with a cluster of observations at one side will

be biased towards this cluster and neglect the locations towards the other directions. Similar as the quadrant search in the pure spatial case an octant wise search strategy for the local neighbourhood would solve this limitation. A simpler stepwise approach to define an n-dimensional neighbourhood might already be sufficient in which at first n_s spatial neighbours and then from each spatial neighbour n_t time instances are selected, such that $n_s \cdot n_t \approx n$.

The presented example considers stationary random fields that are isotropic in space. Further extensions towards more sophisticated variogram estimations allowing also for spatial geometric anisotropy are desirable. One could for instance plot variogram maps for spatial separation in North and South direction for each temporal lag. However, the current implementation does not allow to use the anisotropy parameter `anis` of the pure spatial variogram definition. Nevertheless, a preliminary rescaling of coordinates would be a possible workaround. This route has for instance been taken by Gasch et al. (2015) performing 3D + T kriging. The soil profiles in their study show a clear difference in horizontal and vertical variography. To correct for this, the depth dimension of the data has been rescaled to correspond with the dimensions of the horizontal distances beforehand. In the subsequent study, these pseudo 3D coordinates have been used to fit the spatio-temporal variograms and perform kriging.

The code in model definitions is meant to be kept both flexible and simple. This is based on i) re-producing the notion of the geostatistical models in the R code and in ii) reusing existing definitions and functions of the pure spatial cases that have been available for many years in **gstat**. The data handling benefits to a large degree from the implementations in the **spacetime** R package.

Conclusions

The spatio-temporal extensions to **gstat** allow to model a set of spatio-temporal covariance functions. The implemented functionality eases estimation, visualisation and understanding of spatio-temporal covariance functions. The extension and reuse of already available function structures and nomenclature facilitates an easy translation of spatial workflows to handle spatio-temporal data. The numerical estimation of the variogram parameters might be tricky and needs a large degree of the user's attention. It is advised to carefully check the outcome of the `optim` routine after optimisation. Spatio-temporal kriging predictions can be made in a global and a local neighbourhood set-up, while the latter will be the preferred solution for most spatio-temporal data sets and common computer hardware configurations.

Spatio-temporal covariance structures carry valuable information, but a spatio-temporal model is not guaranteed to outperform pure spatial predictions. The benefit in terms of prediction quality of spatio-temporal kriging becomes only apparent if sufficiently strong correlated locations are added with the temporal dimension (i.e. if the model permits strong correlation across time). Nevertheless, the spatio-temporal covariance model might be of interest in itself.

Besides some publications where the authors of this paper were involved in, such as Kilibarda et al. (2014), the software presented here has proven useful in several independent publications, examples of which are Marek et al. (2015); Biondi (2013); Hu et al. (2015); Yoon et al. (2014).

Acknowledgements

This research has partly been funded by the German Research Foundation (DFG) under project number PE 1632/4-1. We thank two anonymous reviewers for their valuable comments.

Bibliography

K. S. Bakar and S. K. Sahu. spTimer: Spatio-temporal Bayesian modelling using R. *Journal of Statistical Software*, 63(15):1–32, 2015. doi: 10.18637/jss.v063.i15. [p205]

A. Beygelzimer, S. Kakadet, J. Langford, S. Arya, D. Mount, and S. Li. *FNN: Fast Nearest Neighbor Search Algorithms and Applications*, 2013. URL `https://CRAN.R-project.org/package=FNN`. R package version 1.1. [p208]

R. A. Bilonick. Monthly hydrogen ion deposition maps for the northeastern U.S. from July 1982 to September 1984. *Atmospheric Environment (1967)*, 22(9):1909–1924, 1988. doi: 10.1016/0004-6981(88)90080-7. [p206]

F. Biondi. Space-time kriging extension of precipitation variability at 12 km spacing from tree-ring chronologies and its implications for drought analysis. *Hydrology and Earth System Sciences Discussussions*, 10:4301–4335, 2013. doi: 10.5194/hessd-10-4301-2013. [p216]

N. Cressie and C. K. Wikle. *Statistics for Spatio-Temporal Data*. Wiley, 2011. [p204, 205]

L. De Cesare, D. Myers, and D. Posa. Estimating and modeling space-time correlation structures. *Statistics & Probability Letters*, 51(1):9–14, 2001. doi: 10.1016/S0167-7152(00)00131-0. [p205]

S. De Iaco, D. Myers, and D. Posa. Space-time analysis using a general product-sum model. *Statistics & Probability Letters*, 52(1):21–28, 2001. doi: 10.1016/S0167-7152(00)00200-5. [p205, 206]

A. O. Finley, S. Banerjee, and A. E. Gelfand. spBayes for large univariate and multivariate point-referenced spatio-temporal data models. *Journal of Statistical Software*, 63(13), 2015. doi: 10.18637/jss.v063.i13. [p205]

C. K. Gasch, T. Hengl, B. Gräler, H. Meyer, T. S. Magney, and D. J. Brown. Spatio-temporal interpolation of soil water, temperature, and electrical conductivity in 3D + T: The Cook agronomy farm data set. *Spatial Statistics*, 14(Part A):70–90, 2015. doi: 10.1016/j.spasta.2015.04.001. [p216]

Y. Hu, R. Li, R. Bergquist, H. Lynn, F. Gao, Q. Wang, S. Zhang, L. Sun, Z. Zhang, and Q. Jiang. Spatio-temporal transmission and environmental determinants of schistosomiasis japonica in Anhui province, China. *PLoS Neglected Tropical Diseases*, 9(2), 2015. doi: 10.1371/journal.pntd.0003470. [p216]

M. Kilibarda, T. Hengl, G. B. M. Heuvelink, B. Gräler, E. Pebesma, M. Perčec Tadić, and B. Bajat. Spatio-temporal interpolation of daily temperatures for global land areas at 1 km resolution. *Journal of Geophysical Research: Atmospheres*, 119(5):2294–2313, 2014. doi: 10.1002/2013JD020803. [p216]

P. C. Kyriakidis and A. G. Journel. Geostatistical space-time models: A review. *Mathematical Geology*, 31(6):651–684, 1999. doi: 10.1023/A:1007528426688. [p215]

F. Lindgren and H. Rue. Bayesian spatial modelling with R-INLA. *Journal of Statistical Software*, 63(19), 2015. doi: 10.18637/jss.v063.i19. [p205]

L. Marek, P. Tuček, and V. Pászto. Using geovisual analytics in Google Earth to understand disease distribution: A case study of campylobacteriosis in the Czech Republic (2008–2012). *International Journal of Health Geographics*, 14(7):1–13, 2015. URL http://www.ij-healthgeographics.com/content/14/1/7. [p216]

J. C. Nash. On best practice optimization methods in R. *Journal of Statistical Software*, 60(2):1–14, 2014. doi: 10.18637/jss.v060.i02. [p215]

E. Pebesma. spacetime: Spatio-temporal data in R. *Journal of Statistical Software*, 51(7):1–30, 2012. doi: 10.18637/jss.v051.i07. [p204, 205]

E. J. Pebesma. Multivariable geostatistics in S: The gstat package. *Computers & Geosciences*, 30:683–691, 2004. doi: 10.1016/j.cageo.2004.03.012. [p204]

E. J. Pebesma and R. N. Duin. Spatio-temporal mapping of sea floor sediment pollution in the North Sea. In P. Renard and R. Froidevaux, editors, *Fifth European Conference on Geostatistics for Environmental Applications, GeoENV2004*, pages 367–378. Springer, 2005. doi: 10.1007/3-540-26535-X_31. [p215]

E. J. Pebesma, R. N. Duin, and P. A. Burrough. Mapping sea bird densities over the North Sea: Spatially aggregated estimates and temporal changes. *Environmetrics*, 16(6):573–587, 2005. doi: 10.1002/env.723. [p215]

M. Schlather, A. Malinowski, P. J. Menck, M. Oesting, and K. Strokorb. Analysis, simulation and prediction of multivariate random fields with package RandomFields. *Journal of Statistical Software*, 63(8):1–25, 2014. doi: 10.18637/jss.v063.i08. [p204]

F. Sigrist, H. R. Künsch, and W. A. Stahel. spate: An R package for spatio-temporal modeling with a stochastic advection-diffusion process. *Journal of Statistical Software*, 63(14):1–23, 2015. doi: 10.18637/jss.v063.i14. [p205]

J. Snepvangers, G. Heuvelink, and J. Huisman. Soil water content interpolation using spatio-temporal kriging with external drift. *Geoderma*, 112(3–4):253–271, 2003. doi: 10.1016/S0016-7061(02)00310-5. [p206]

R. Varadhan. Numerical optimization in R: Beyond optim. *Journal of Statistical Software*, 60(1):1–3, 9 2014. doi: 10.18637/jss.v060.i01. [p215]

S. Y. Yoon, S. K. Ravulaparthy, and K. G. Goulias. Dynamic diurnal social taxonomy of urban environments using data from a geocoded time use activity-travel diary and point-based business establishment inventory. *Transportation Research Part A: Policy and Practice*, 68:3–17, 2014. doi: 10.1016/j.tra.2014.01.004. [p216]

Benedikt Gräler
Institute for Geoinformatics, University of Münster
Heisenbergstr. 2, 48149, Münster
Germany
ben.graeler@uni-muenster.de

Edzer Pebesma
Institute for Geoinformatics, University of Münster
Heisenbergstr. 2, 48149, Münster
Germany
edzer.pebesma@uni-muenster.de

Gerard Heuvelink
Department of Environmental Sciences, Wageningen University
PO Box 47, 6700AA, Wageningen
The Netherlands
gerard.heuvelink@wur.nl

Appendix

Derivation of the separable covariance and variogram identities

The separable covariance and variogram identity is readily available through

$$\begin{aligned}
C_{\text{sep}}(h,u) &= C_s(h)C_t(u) = \text{sill} \cdot \bar{c}_s(h)\bar{c}_t(u) \\
\gamma_{\text{sep}}(h,u) &= C_{\text{sep}}(0,0) - C_{\text{sep}}(h,u) \\
&= \text{sill}\,(1 - \bar{c}_s(h) \cdot \bar{c}_t(u)) \\
&= \text{sill}\,(1 - (1 - \bar{\gamma}_s(h))\,(1 - \bar{\gamma}_t(u))) \\
&= \text{sill}\,(1 - (1 - \bar{\gamma}_s(h) - \bar{\gamma}_t(u) + \bar{\gamma}_s(h)\bar{\gamma}_t(u))) \\
&= \text{sill}\,(\bar{\gamma}_s(h) + \bar{\gamma}_t(u) - \bar{\gamma}_s(h)\bar{\gamma}_t(u))
\end{aligned}$$

where $\bar{c}$ and $\bar{\gamma}$ are normalised correlation and correlogram functions respectively.

Derivation of the product-sum covariance and variogram identities

The product-sum covariance and variogram identity is readily available through:

$$\begin{aligned}
C_{\text{ps}}(h,u) &= k \cdot C_s(h)C_t(u) + C_s(h) + C_t(u) \\
\gamma_{\text{ps}}(h,u) &= C_{\text{ps}}(0,0) - C_{\text{ps}}(h,u) \\
&= k \cdot C_s(0)C_t(0) + C_s(0) + C_t(0) \\
&\quad - (k \cdot C_s(h)C_t(u) + C_s(h) + C_t(u)) \\
&= k \cdot \text{sill}_s \cdot \text{sill}_t + \text{sill}_s + \text{sill}_t \\
&\quad - k \cdot [(\text{sill}_s - \gamma_s(h))\,(\text{sill}_t - \gamma_t(u))] - (\text{sill}_s - \gamma_s(h)) - (\text{sill}_t - \gamma_t(u)) \\
&= k \cdot \text{sill}_s \cdot \text{sill}_t + \text{sill}_s + \text{sill}_t \\
&\quad - k \cdot [\text{sill}_s \cdot \text{sill}_t - \text{sill}_s \cdot \gamma_t(u) - \text{sill}_t \cdot \gamma_s(h) + \gamma_s(h)\gamma_t(u)] \\
&\quad - \text{sill}_s + \gamma_s(h) - \text{sill}_t + \gamma_t(u) \\
&= k \cdot \text{sill}_t\gamma_s(h) + k \cdot \text{sill}_s\gamma_t(u) - k\gamma_s(h)\gamma_t(u) + \gamma_s(h) + \gamma_t(u) \\
&= (k \cdot \text{sill}_t + 1)\gamma_s(h) + (k \cdot \text{sill}_s + 1)\gamma_t(u) - k\gamma_s(h)\gamma_t(u)
\end{aligned}$$

SWMPr: An R Package for Retrieving, Organizing, and Analyzing Environmental Data for Estuaries

by Marcus W Beck

Abstract The System-Wide Monitoring Program (SWMP) was implemented in 1995 by the US National Estuarine Research Reserve System. This program has provided two decades of continuous monitoring data at over 140 fixed stations in 28 estuaries. However, the increasing quantity of data provided by the monitoring network has complicated broad-scale comparisons between systems and, in some cases, prevented simple trend analysis of water quality parameters at individual sites. This article describes the **SWMPr** package that provides several functions that facilitate data retrieval, organization, and analysis of time series data in the reserve estuaries. Previously unavailable functions for estuaries are also provided to estimate rates of ecosystem metabolism using the open-water method. The **SWMPr** package has facilitated a cross-reserve comparison of water quality trends and links quantitative information with analysis tools that have use for more generic applications to environmental time series.

Introduction

The development of low-cost, automated sensors that collect data in near real time has enabled a proliferation of standardized environmental monitoring programs (Glasgow et al., 2004; Fries et al., 2008). An invaluable source of monitoring data for coastal regions in the United States is provided by the National Estuarine Research Reserve System (NERRS, http://www.nerrs.noaa.gov/). This network of 28 estuary reserves was created to address long-term research, monitoring, education, and stewardship goals in support of coastal management. The System-Wide Monitoring Program (SWMP) was implemented in 1995 at over 140 stations across the reserves to provide a robust, long-term monitoring system for water quality, weather, and land-use/habitat change. Environmental researchers have expressed a need for quantitative analysis tools to evaluate trends in water quality time series given the quantity and quality of data provided by SWMP (System-Wide Monitoring Program Data Analysis Training, 2014).

This article describes the **SWMPr** package that was developed for estuary monitoring data from the SWMP. Functions provided by **SWMPr** address many common issues working with large datasets created from automated sensor networks, such as data pre-processing to remove unwanted information, combining data from different sources, and exploratory analyses to identify parameters of interest. Additionally, web applications derived from **SWMPr** and **shiny** illustrate potential applications using the functions in this package. The software is provided specifically for use with NERRS data, although many of the applications are relevant for addressing common challenges working with large environmental datasets.

Overview of the SWMP network

The **SWMPr** package was developed for the continuous abiotic monitoring network that represents a majority of SWMP data and, consequently, the most challenging to evaluate. Abiotic elements monitored at each reserve include water quality (water temperature, specific conductivity, salinity, dissolved oxygen concentration, dissolved oxygen saturation, depth, pH, turbidity, chlorophyll fluorescence), weather (air temperature, relative humidity, barometric pressure, wind speed, wind direction, photosynthetically active radiation, precipitation), and nutrient data (orthophosphate, ammonium, nitrite, nitrate, nitrite + nitrate, chlorophyll a). Each of the 28 estuary reserves has no fewer than four water quality stations and one weather station at fixed locations. Water quality and weather data are collected at 15 minute intervals, whereas nutrient data are collected monthly at each water quality station. Data are made available through the Centralized Data Management Office (CDMO) web portal (http://cdmo.baruch.sc.edu/), where quality assurance/quality control (QAQC) measures are used to screen the information for accuracy and reliability. The final data include timestamped observations with relevant QAQC flags.

At the time of writing, the CDMO web portal provides over 60 million water quality, weather, and nutrient records that have been authenticated through systematic QAQC procedures. Records for each station are identified by a seven or eight character name that specifies the reserve, station, and

Table 1: Retrieval functions available from the **SWMPr** package. Full documentation for each function is in the help file (e.g., execute `?all_params` for individual functions or `help.search('retrieve', package = 'SWMPr')` for all).

Function	Description
`all_params`	Retrieve records starting with the most recent at a given station, all parameters. Wrapper to `exportAllParamsXMLNew` function on web services.
`all_params_dtrng`	Retrieve records of all parameters within a given date range for a station. Optional argument for a single parameter. Wrapper to `exportAllParamsDateRangeXMLNew`.
`import_local`	Import files from a local path. The files must be in a specific format, such as those returned from the CDMO using the zip downloads option.
`single_param`	Retrieve records for a single parameter starting with the most recent at a given station. Wrapper to `exportSingleParamXMLNew` function on web services.
`site_codes`	Get metadata for all stations. Wrapper to `exportStationCodesXMLNew` function on web services.
`site_codes_ind`	Get metadata for all stations at a single site. Wrapper to `NERRFilterStationCodesXMLNew` function on web services.

parameter type. For example, 'apaebwq' is the water quality identifier ('wq') for the East Bay station ('eb') at the Apalachicola reserve ('apa'). Similarly, a suffix of 'met' or 'nut' specifies the weather (meteorological) or nutrient stations. All reserve names, stations, and date ranges for each parameter type can be viewed on the CDMO website. Alternatively, the `site_codes` (all sites) or `site_codes_ind` (single site) functions provided by **SWMPr** can be used. As noted below, an IP address must be registered with CDMO before using the data retrieval functions in **SWMPr**. Web services are provided by CDMO for direct access to SWMP data through http requests, in addition to standard graphical user interface options for selecting data. The data retrieval functions in **SWMPr** are simple calls to the existing retrieval functions on CDMO web services, as explained below.

Structure of the SWMPr package

SWMPr functions are categorized by one of three steps in the data workflow: ***retrieving***, ***organizing***, and ***analyzing***. Functions for retrieving are used to import the data into R as a `"swmpr"` object class. Functions for organizing and analyzing the data provide methods for working with a `"swmpr"` object. The following describes the package structure, beginning with the retrieval functions, a description of the `"swmpr"` object returned after retrieval, and, finally, the organizing and analyzing functions.

Data retrieval

SWMPr can import data into R through direct download from the CDMO or by importing local data that was previously downloaded (Table 1). The IP address for the computer making the request must be registered if the first approach is used (see CDMO website). The `site_codes` or `site_codes_ind` functions can be used to view site metadata.

```
# retrieve metadata for all sites
site_codes()

# retrieve metadata for a single site
site_codes_ind('apa')
```

Retrieval functions to import data directly into R from the CDMO include `all_params`, `all_params_dtrng`, and `single_param`. Due to rate limitations on the CDMO server, the retrieval functions return a limited number of records with each request. However, the **SWMPr** functions use the native CDMO web services iteratively (i.e., within a loop) to obtain all requested records. Download time can be excessive for longer time series.

```
# all parameters for a station, most recent
all_params('hudscwq')

# get all parameters within a date range
all_params_dtrng('hudscwq', dtrng = c('09/01/2013', '10/01/2013'))

# get single parameter within a date range
all_params_dtrng('hudscwq', dtrng = c('09/01/2013', '10/01/2013'),
  param = 'do_mgl')

# single parameter for a station, most recent
single_param('hudscwq', param = 'do_mgl')
```

The second approach for data retrieval is to use the `import_local` function to import data into R after downloading from CDMO. This approach is most appropriate for large data requests. The `import_local` function is designed for data from the zip downloads feature in the advanced query section of the CDMO website. The zip downloads feature can be used to obtain a large number of records from multiple stations in one request. The downloaded data will be in a compressed folder that includes multiple .csv files by year for a given data type (e.g., apacpwq2002.csv, apacpwq2003.csv, apacpnut2002.csv, etc.). The `import_local` function can be used to import files directly from the zipped folder.

The "swmpr" object class

All data retrieval functions return a "swmpr" object that includes relevant data and several attributes describing the dataset. The data include a `datetimestamp` column in the timezone for a station and additional parameters for the data type (weather, nutrients, or water quality). Corresponding QAQC columns for each parameter are also returned if provided by the initial data request. The following shows an example of the raw data imported using `all_params`.

```
# import all paramaters for the station
# three most recent records
exdat <- all_params('apadbwq', Max = 3, trace = F)
exdat
##         datetimestamp temp f_temp spcond f_spcond sal f_sal do_pct
## 1 2015-11-03 11:15:00   26      0     45        0  29     0     78
## 2 2015-11-03 11:30:00   26      0     46        0  30     0     76
## 3 2015-11-03 11:45:00   26      0     46        0  30     0     75
##   f_do_pct do_mgl f_do_mgl depth f_depth ph f_ph turb f_turb chlfluor
## 1        0      5        0     2       0  8    0    2      0       NA
## 2        0      5        0     2       0  8    0    5      0       NA
## 3        0      5        0     2       0  8    0    5      0       NA
##   f_chlfluor level f_level cdepth clevel f_cdepth f_clevel
## 1         -2    NA      -1      2     NA        3
## 2         -2    NA      -1      2     NA        3
## 3         -2    NA      -1      2     NA        3
```

The attributes of a "swmpr" object are descriptors that are appended to the raw data (Table 2). These act as metadata that are used internally by many of the package functions and are updated as the data are processed. The attributes are not visible with the raw data but can be viewed as follows.

```
# import sample data from package
data(apadbwq)
dat <- apadbwq

# view all attributes of dat
attributes(dat)

# view a single attribute of dat
attr(dat, 'station')
```

The "swmpr" object class was created for use with the organizing and analyzing functions. This uses the standard S3 object class system for R, such that specific methods for generic functions are developed for the object class. A "swmpr" object also secondarily inherits methods from the "data.frame" class. Available methods for the "swmpr" class are described below and can also be viewed:

Table 2: Attributes of a "swmpr" object that describe characteristics of the data.

Attributes	Class	Description
names	character	Column names of the entire data set, inherited from the data.frame object class.
row.names	integer	Row names of the data set, inherited from the data.frame object class.
class	character	Class of the data object indicating "swmpr" and "data.frame".
station	character	Station identifier used by NERRS as a string with 7 or 8 characters.
parameters	character	Character vector of column names for data parameters, e.g., 'do_mgl', 'turb', etc.
qaqc_cols	logical	Indicates if QAQC columns are present in the raw data.
date_rng	POSIXct	Start and end dates for the data.
timezone	character	Timezone of the station using the city/country format[a].
stamp_class	character	Class of the datetimestamp column, usually "POSIXct" unless data have been aggregated.

[a]Time zones that do not observe daylight savings are used for "swmpr" objects and may not be cities in the United States. For example, America/Jamaica is used for Eastern Standard Time.

```
# view available methods for swmpr class
methods(class = 'swmpr')
```

Data organizing

The organize functions are used to 'clean' or prepare the imported data for analysis, including viewing and removal of QAQC flags, subsetting, combining replicate nutrient observations, creating a standardized time series, and combining data of different types (Table 3).

The qaqc function is a simple screen to retain observations from the data with specified QAQC flags (see http://cdmo.baruch.sc.edu/data/qaqc.cfm). Each parameter in the imported "swmpr" object will have a corresponding QAQC column of the same name with the added prefix f_ (e.g., f_do_mgl for do_mgl). Values in the QAQC column range from -5 to 5 to indicate the QAQC flag that was assigned by CDMO during initial processing. The qaqc function is used to remove observations in the raw data with given flags, with the default option to retain only values with the 0 QAQC flag (i.e., passed initial CDMO checks). Additionally, simple filters are used to remove obviously bad values (e.g., wind speed values less than zero or pH values greater than 12). Erroneous data entered as -99 are also removed. The function returns the original data with the QAQC columns removed and NA (not available) values for observations that do not meet the criteria specified in the function call.

```
# qaqc screen for a swmpr object, retain only '0'
qaqc(dat)

# retain all data regardless of flag
qaqc(dat, qaqc_keep = NULL)

# retain only '0' and '-1' flags
qaqc(dat, qaqc_keep = c(0, -1))
```

SWMP data often contain observations above or below the detection limit for the sensor or laboratory method used to quantify the parameters. For example, nutrient data exceeding the high sensor range are assigned a QAQC flag of -5, whereas data below the low sensor range are assigned a QAQC flag of -4. The presence of censored data is non-trivial and can influence the types of analyses that are appropriate for a time series (Helsel, 2012). A detailed discussion of methods for evaluating censored data is beyond the scope of the manuscript and existing methods for R are provided by other packages (e.g., **cents**, McLeod et al., 2014). However, the functions in **SWMPr** can be used to identify censored data based on the appropriate QAQC flag for a parameter. Viewing this information can be

Table 3: Organizing functions available from the **SWMPr** package. Full documentation for each function is in the help file (e.g., execute `?comb` for individual functions or `help.search('organize', package = 'SWMPr')` for all).

Function	Description
comb	Combines "swmpr" objects to a common time series using setstep, such as combining the weather, nutrients, and water quality data for a single station.
qaqc	Remove QAQC columns and remove data based on QAQC flag values for a "swmpr" object.
qaqcchk	View a summary of the number of observations in a "swmpr" object that are assigned to each QAQC flag used by CDMO.
rem_reps	Remove replicate nutrient data that occur on the same day. The default is to average replicates.
setstep	Format data from a "swmpr" object to a continuous time series at a given timestep.
subset	Subset by dates and/or columns for a "swmpr" object. This is a method passed to the generic subset function in the base installation.

helpful for determining how to further process the data with the qaqc function or alternative methods outside of **SWMPr**. The qaqcchk function returns a data.frame of the number of observations for a parameter that is assigned to all QAQC flags, including those for censored data. SWMP data should not be analyzed without viewing this information to determine an appropriate method to address data with questionable QAQC flags.

```
# view the number of observations in each QAQC flag
qaqcchk(dat)
```

A subset method added to the existing generic subset function in R is available for "swmpr" objects. This function is used to subset the data by date and/or a selected parameter. The date can be a single value or as two dates to select records within the range. The former case requires a binary operator as a character string passed to the operator argument, such as '>' or '<='. The subset argument for the date(s) must also be a character string of the format YYYY-mm-dd HH:MM for each element (e.g., '2007-01-01 06:30').

```
# import data
data(apaebmet)
dat <- apaebmet

# select two parameters from dat
subset(dat, select = c('rh', 'bp'))

# subset records greater than or equal to a date
subset(dat, subset = '2013-01-01 0:00', operator = '>=')

# subset records within a date range, select two parameters
subset(dat, subset = c('2012-07-01 6:00', '2012-08-01 18:15'),
  select = c('atemp', 'totsorad'))
```

The setstep function formats a "swmpr" object to a continuous time series at a given time step. The function also has a default method making it useful for standardizing arbitrary time series to a given interval. The first argument of the function, timestep, specifies the desired time step in minutes starting from the nearest hour of the first observation. The second argument, differ, specifies the allowable tolerance in minutes for matching existing observations to the defined time steps in cases where the two are dissimilar. Values for differ that are greater than one half of the value of timestep are not allowed to prevent duplication of existing data. Likewise, the default value for differ is one half of the time step.

```
# import, qaqc removal
data(apadbwq)
dat <- qaqc(apadbwq)

# convert time series to two hour invervals
# tolerance of +/- 30 minutes for matching existing data
setstep(dat, timestep = 120, differ = 30)
```

The comb function is used to combine multiple "swmpr" objects into a single object with a continuous time series at a given step. The setstep function is used internally such that timestep and differ are accepted arguments for comb. Data are combined by creating a master time series that is used to iteratively merge all "swmpr" objects. The time series for merging depends on the value passed to the method argument. Passing 'union' to method will create a time series that is continuous from the earliest and latest dates for all input objects, whereas 'intersect' will create a continuous time series from the set of dates that are shared between input objects. A character string or numeric vector can also be used to specify which of the input objects to use as the master time series for combining. As with setstep, a default method for comb is provided to allow use with arbitrary data structures. Both functions treat missing data as NA values, either for observations that exceed the allowable tolerance for the differ argument of setstep or for portions of time series that do not overlap given the method argument passed to comb.

```
# get nut, wq, and met data as separate objects
data(apacpnut)
data(apacpwq)
data(apaebmet)
swmp1 <- apacpnut
swmp2 <- apacpwq
swmp3 <- apaebmet

# combine nut and wq data by union
comb(swmp1, swmp2, method = 'union')

# combine nut and met data by intersect
comb(swmp1, swmp3, method = 'intersect')

# combine nut, wq, and met data by nut time series, two hour time step
comb(swmp1, swmp2, swmp3, timestep = 120, method = 'apacpnut')
```

Data analysis

The analysis functions range from general purpose tools for time series analysis to more specific functions for working with continuous monitoring data in estuaries (Table 4). The general purpose tools are "swmpr" methods for existing S3 generics or are slight modifications to existing functions. These include aggreswmp to combine observations by set periods of time (e.g., weeks, months), smoother to average time series with a moving window, and approx to substitute missing data with interpolated values. For brevity, the general functions are not discussed. More specific functions for environmental time series include decomposition functions, decomp and decomp_cj, and functions to estimate and plot ecosystem metabolism from combined water quality and weather data. Several plotting methods to facilitate analysis are also descibed below.

The disaggregation of time series into additive or multiplicative components is a common application for trend analysis. The decomp function is a simple wrapper to decompose (Kendall and Stuart, 1983) that separates a time series into a trend, cyclical variation (e.g., daily or annual), and the remainder (Figure 1). An additive decomposition assumes that the cyclical component of the time series is stationary (i.e., the variance is constant); otherwise, a multiplicative decomposition can be used. The frequency argument describes the periodicity of the cyclical parameter in units of the native time step. For example, the frequency for a parameter with daily periodicity would be 96 if the time step is 15 minutes (24 hours * 60 minutes / 15 minutes). For simplicity, character strings of 'daily' or 'annual' can be supplied in place of numeric values, although any number can be used to identify an arbitrary cyclical component. A starting value of the time series must be supplied in the latter case that indicates the sequence in the cycle for the first observation (see ts for details).

```
# get data
data(apadbwq)
dat <- apadbwq
```

Table 4: Analysis functions available from the **SWMPr** package. Full documentation for each function is in the help file (e.g., execute `?aggreswmp` for individual functions or `help.search('analyze', package = 'SWMPr')` for all).

Function	Description
`aggreswmp`	Aggregate `"swmpr"` objects for different time periods - years, quarters, months, weeks, days, or hours. The aggregation function defaults to the mean.
`aggremetab`	Aggregate metabolism data from a `"swmpr"` object. This is primarly used within `plot_metab` but may be useful for simple summaries of daily metabolism data.
`ecometab`	Estimate ecosystem metabolism for a combined water quality and weather dataset using the open-water method (Odum, 1956).
`decomp`	Decompose a `"swmpr"` time series into trend, seasonal, and residual components. This is a simple wrapper to `decompose` (Kendall and Stuart, 1983). Decomposition of monthly or daily trends is possible.
`decomp_cj`	Decompose a `"swmpr"` time series into grandmean, annual, seasonal, and events components. This is a simple wrapper to `decompTs` in the **wq** package (Jassby and Cloern, 2014). Only monthly decomposition is possible.
`hist`	Plot a histogram for a single variable.
`lines`	Add lines to an existing plot created with `plot`.
`map_reserve`	Create a map of all stations in a reserve using the **ggmap** package (Kahle and Wickham, 2013).
`na.approx`	Linearly interpolate missing data (`NA` values) in a `"swmpr"` object.
`overplot`	Plot multiple time series in a `"swmpr"` object on the same y-axis.
`plot`	Plot a univariate time series for a `"swmpr"` object.
`plot_metab`	Plot ecosystem metabolism estimates after running `ecometab` on a combined `"swmpr"` object.
`plot_summary`	Create summary plots of seasonal/annual trends and anomalies for a single parameter.
`smoother`	Smooth `"swmpr"` objects with a moving window average, passed to `filter`.

```
# subset for daily decomposition
dat <- subset(dat, subset = c('2013-07-01 00:00', '2013-07-31 00:00'))

# daily decomposition of DO and plot
dc_dat <- decomp(dat, param = 'do_mgl', frequency = 'daily')
plot(dc_dat)
```

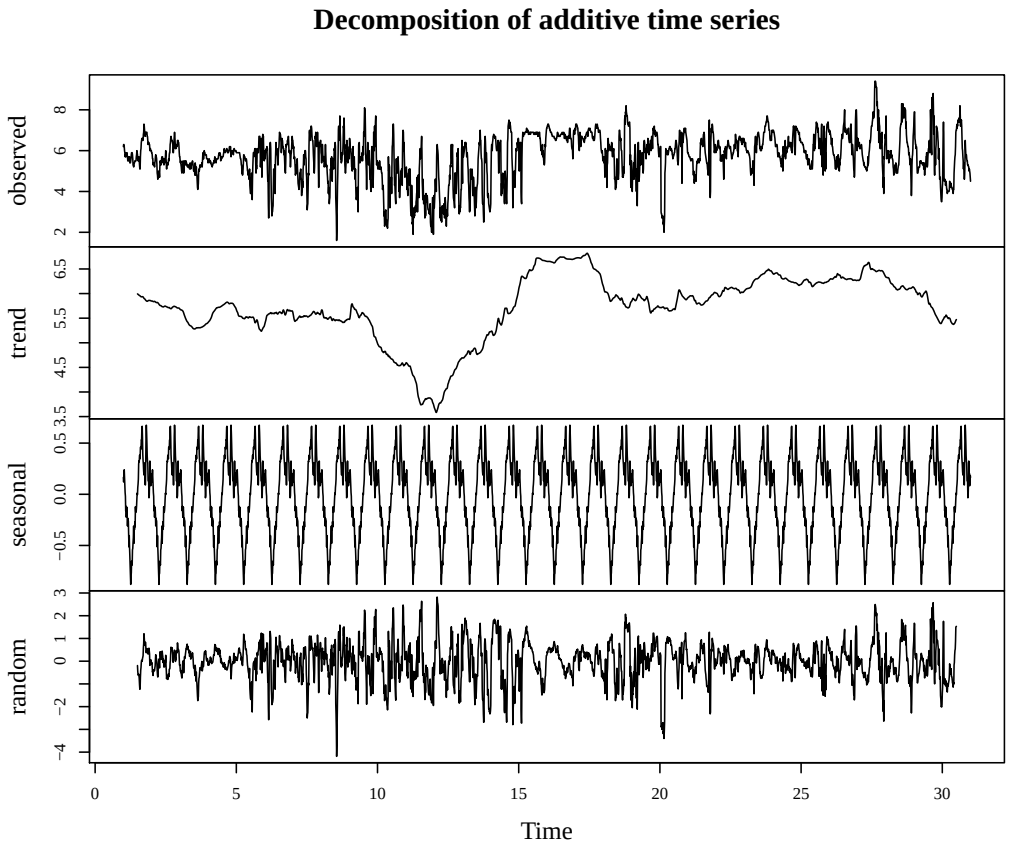


Figure 1: An additive decomposition of dissolved oxygen into a trend, seasonal (daily), and random component using the `decomp` function.

An alternative approach for decomposition is provided by the `decomp_cj` function, which is a simple wrapper to the `decompTs` function in the **wq** package (Cloern and Jassby, 2010; Jassby and Cloern, 2014). The `decomp_cj` function is a monthly decomposition for characterizing relatively long-term trends. This approach works best for nutrient data that are typically obtained on a monthly cycle. The time series is decomposed into the grandmean, annual, seasonal, and events components (Figure 2), as compared to trend, seasonal, and random components for the `decomp` function above. For both functions, the random or events components can be considered anomalies that do not follow the trends in the remaining categories. Additional arguments passed to `decompTs` can be used with `decomp_cj`, such as `startyr`, `endyr`, and `type`. Values passed to `type` are `mult` (default) or `add`, referring to multiplicative or additive decomposition.

```
# get data
data(apacpnut)
dat <- apacpnut
dat <- qaqc(dat, qaqc_keep = NULL)

# decomposition of chl
decomp_cj(dat, param = 'chla_n')
```

Estimates of ecosystem metabolism provide a measure of system productivity to evaluate whether an ecosystem is a net source or sink of organic material. The open-water method (Odum, 1956) is a

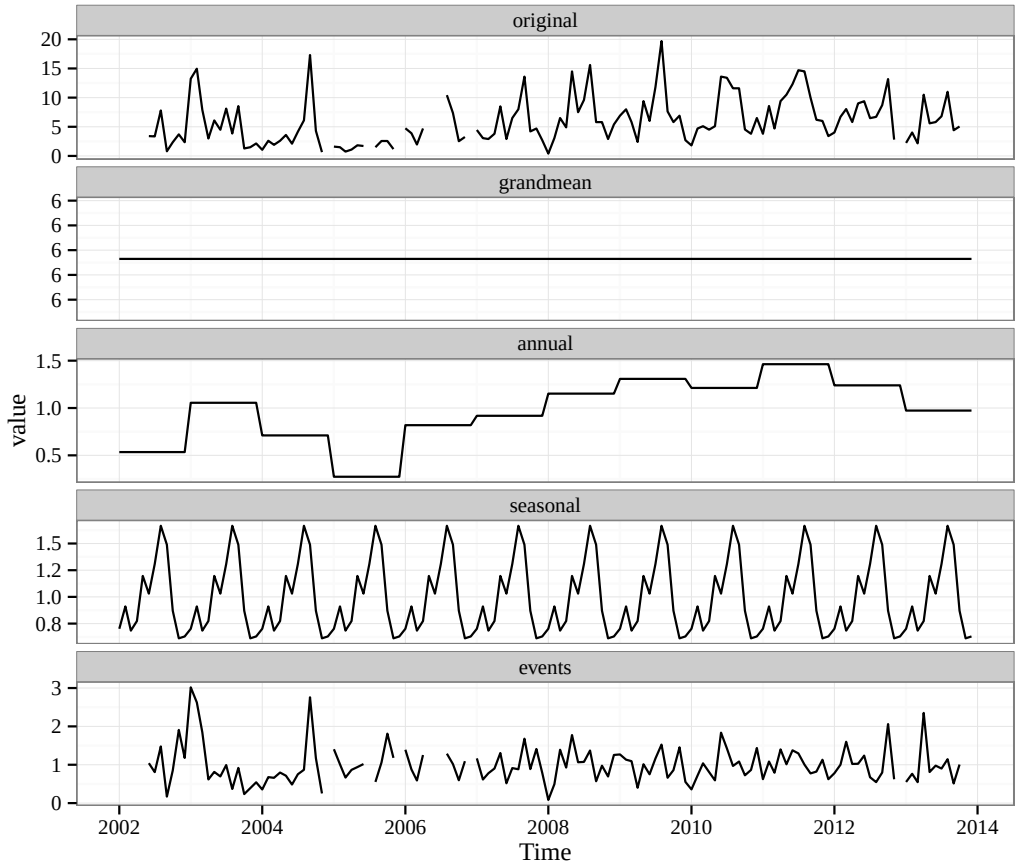


Figure 2: Additive decomposition of a multi-year chlorophyll time series into the grandmean, annual, seasonal, and events components using the `decomp_cj` function.

common approach to quantify metabolism using a mass balance equation that describes the change in dissolved oxygen over time from the balance between photosynthetic and respiration processes, corrected using an empirically constrained air-sea gas diffusion model (Ro and Hunt, 2006; Thébault et al., 2008). A detailed discussion of the method is beyond the scope of this article, although users are encouraged to consult references herein for additional information (see Kemp and Testa (2012); Needoba et al. (2012); Caffrey et al. (2013), also the package help files). Methods for estuaries have not previously been available in R, although the **StreamMetabolism** package provides an approach for freshwater systems. The following is an example that shows use of `ecometab` with a combined water quality and weather data set. Monthly aggregations of the raw, daily estimates are plotted using `plot_metab` (Figure 3).

```
## import water quality and weather data
data(apadbwq)
data(apaebmet)

## qaqc, combine
wq <- qaqc(apadbwq)
met <- qaqc(apaebmet)
dat <- comb(wq, met)

## estimate metabolism
res <- ecometab(dat, trace = FALSE)
plot_metab(res)
```

Exploratory graphics are also useful for evaluating general trends in observed data. Several graphics showing seasonal and annual trends for a single SWMP parameter can be obtained using the `plot_summary` function (Figure 4). The plots include monthly distributions, monthly anomalies, and annual anomalies in multiple formats. An interactive **shiny** web application (Chang et al., 2015) that uses this function is available for viewing results for all SWMP sites (see the Applications using the

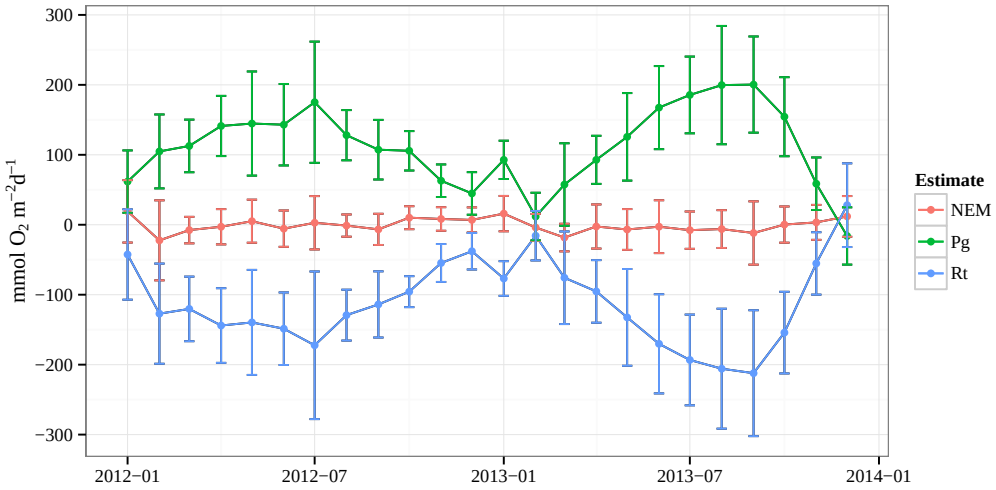


Figure 3: Monthly means (95% confidence) of ecosystem metabolism estimates (net ecosystem metabolism, gross production, and total respiration) for combined water quality and weather data for two years at Apalachicola Bay, Florida.

SWMPr package section).

```
## import data
data(apacpnut)
dat <- qaqc(apacpnut)

## plot
plot_summary(dat, param = 'chla_n', years = c(2007, 2013))
```

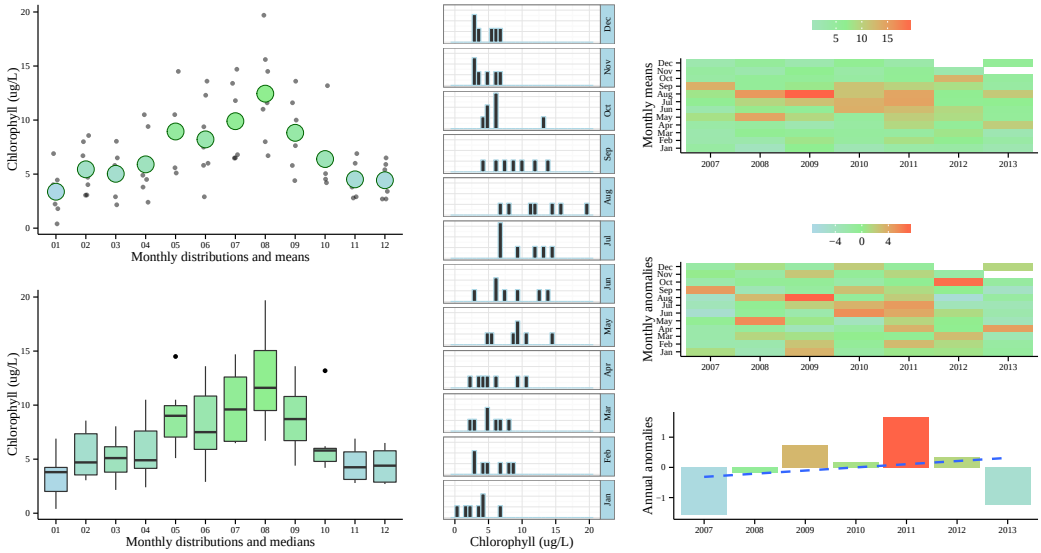


Figure 4: Summaries of a multi-year chlorophyll time series using the `plot_summary` function. Summaries include monthly distributions (means on top left, quantiles on bottom left), monthly histograms (center), monthly means by year (top right), deviation from monthly means (middle right), and annual trends as deviations from the grand mean (bottom right)

Similarly, the `overplot` function provides an alternative approach to viewing observed data from the same station. This function uses the base **graphics** package to plot multiple time series on the same y-axis (Figure 5).

```
## import data
data(apacpwq)
```

```
dat <- qaqc(apacpwq)

## plot
overplot(dat, select = c('depth', 'do_mgl', 'ph', 'turb'),
  subset = c('2013-01-01 0:0', '2013-02-01 0:0'), lwd = 2)
```

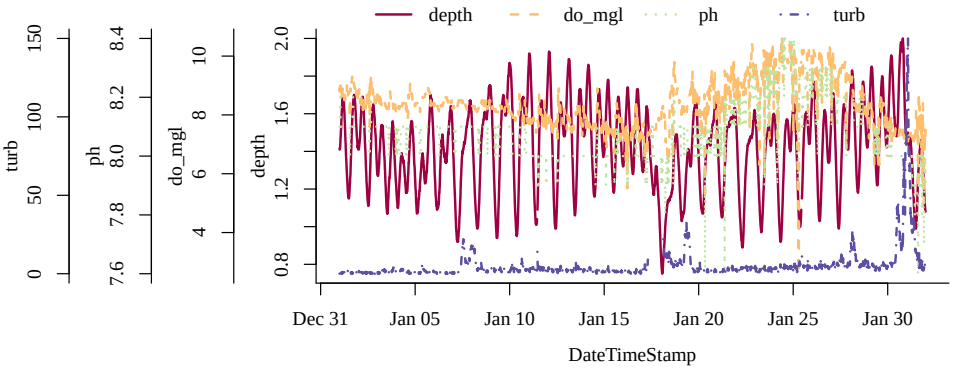


Figure 5: The `overplot` function plots multiple variables on the same y-axis.

Finally, the `map_reserve` function can be used to create a map of stations at a reserve using the **ggmap** package (Figure 6, Kahle and Wickham (2013)). The function uses Google maps of four types that can be set with the `map_type` argument: terrain (default), satellite, roadmap, or hybrid. The `zoom` argument can be chosen through trial and error depending on the spatial extent of the reserve.

```
# plot the stations at Jacques Cousteau reserve
map_reserve('jac')
```

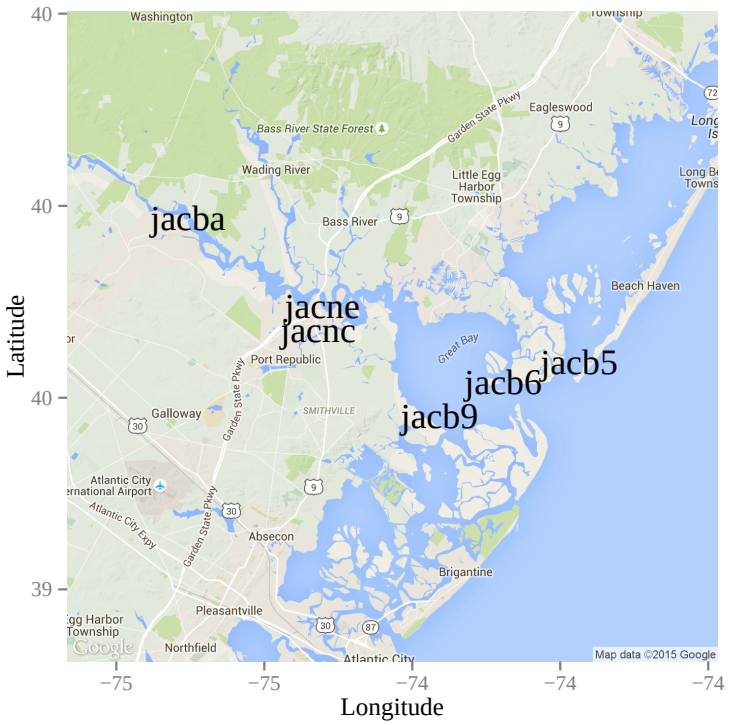


Figure 6: Locations of all sites at the Jacques Cousteau reserve using the `map_reserve` function.

Applications using the SWMPr package

Two **shiny** web applications illustrate the improved ability to synthesize and evaluate multi-year time series using **SWMPr**. The first application evaluates trends in SWMP data within and between sites using an interactive **leaflet** map (Cheng and Xie (2015), Figure 7): `https://beckmw.shinyapps.io/swmp_comp`. Trends between reserves can be viewed using the map, whereas trends at individual sites can be viewed by clicking on a map location. Site-level trends are shown below the map with a simple linear regression to show an increase or decrease in values over time, whereas trends between sites are shown on the map for each station as circles that identify the direction and significance of the trend. More robust methods for evaluating trends are currently not provided by the application and the use of simple linear regression is meant for initial exploratory analysis. The second application provides graphical summaries of water quality, weather, or nutrient station data at individual stations using the `plot_summary` function: `https://beckmw.shinyapps.io/swmp_summary/`. The output is identical to Figure 4 with the addition of drop down menus to select the station, date range, and parameter for plotting.

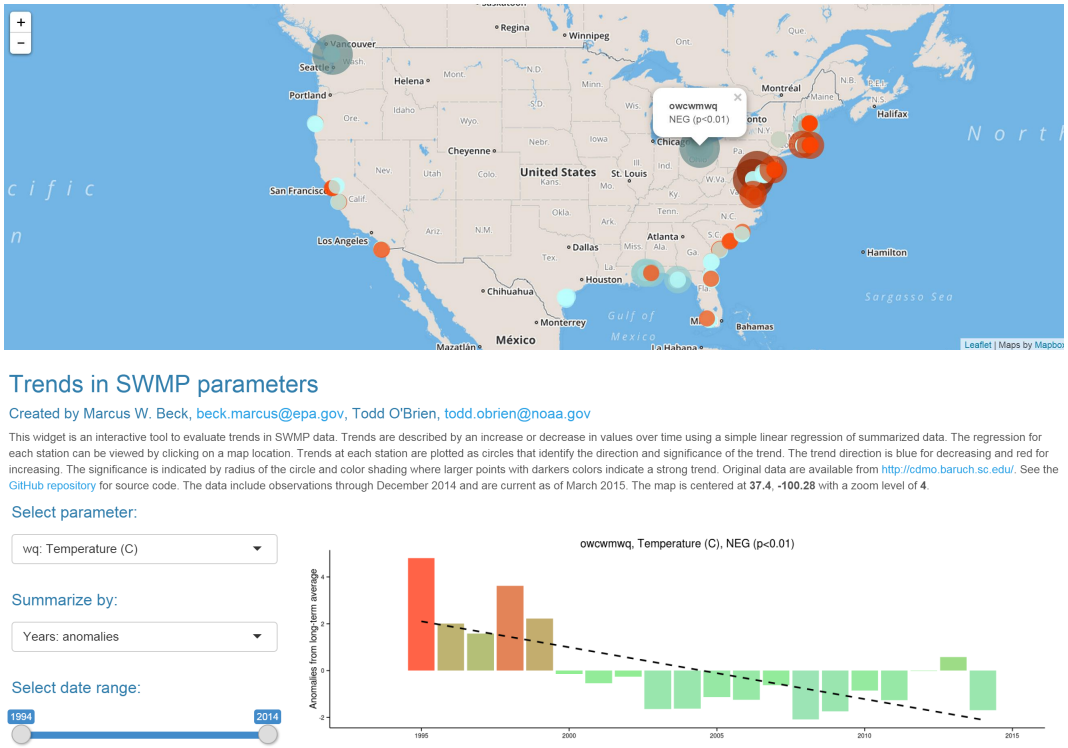


Figure 7: Online application for comparing trends in SWMP data parameters using an interactive map. Link: `https://beckmw.shinyapps.io/swmp_comp`

Conclusions

SWMPr was developed to augment existing data management programs (i.e., CDMO) by providing a bridge betwen the raw data and the analysis software through its numerous data retrieval functions (Table 1). Established QAQC methods and data processing techniques are also enhanced with **SWMPr** by functions that filter observations for different QAQC flags (`qaqc`) and subset by selected dates or variables (`subset`). Additionally, challenges comparing differents datasets are addressed by the `setstep` and `comb` functions that standardize and combine time series. Finally, the analysis functions provide numerous tools to implement common analyses for time series and more specific methods for water quality data. Further development of the package will include modifications and additional functions to better integrate data analysis with the quality of information provided by SWMP. Several functions include default methods to extend use beyond the `"swmpr"` object and additional development will continue to focus on modifying the package to handle arbitrary data structures. These challenges are not unique to the SWMP database such that many of the functions will facilitate evaluations of more generic time series datasets.

Acknowledgments

I acknowledge the significant work of NERRS researchers and staff that has allowed access to high-quality monitoring data. Thanks to Todd O'Brien for the inspiration for the online widgets. Thanks to Mike Murrell and Jim Hagy III for assistance with the ecosystem metabolism functions. Thanks to Jeff Hollister for providing useful comments on an earlier draft.

Bibliography

J. M. Caffrey, M. C. Murrell, K. S. Amacker, J. Harper, S. Phipps, and M. Woodrey. Seasonal and inter-annual patterns in primary production, respiration and net ecosystem metabolism in 3 estuaries in the northeast Gulf of Mexico. *Estuaries and Coasts*, 37(1):222–241, 2013. [p227]

W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson. *shiny: Web Application Framework for R*, 2015. URL http://CRAN.R-project.org/package=shiny. R package version 0.11.1. [p227]

J. Cheng and Y. Xie. *leaflet: Create Interactive Web Maps with the JavaScript 'Leaflet' Library*, 2015. URL http://CRAN.R-project.org/package=leaflet. R package version 1.0.0. [p230]

J. E. Cloern and A. D. Jassby. Patterns and scales of phytoplankton variability in estuarine-coastal ecosystems. *Estuaries and Coasts*, 33(2):230–241, 2010. [p226]

D. P. Fries, S. Z. Ivanov, P. H. Bhanushali, J. A. Wilson, H. A. Broadbent, and A. C. Sanderson. Broadband, low-cost, coastal sensor nets. *Oceanography*, 20(4):150–155, 2008. [p219]

H. B. Glasgow, J. M. Burkholder, R. E. Reed, A. J. Lewitus, and J. E. Kleinman. Real-time remote monitoring of water quality: a review of current applications, and advancements in sensor, telemetry, and computing technologies. *Journal of Experimental Marine Biology and Ecology*, 300(1-2):409–448, 2004. [p219]

D. R. Helsel. *Statistics for Censored Environmental Data Using Minitab and R*. John Wiley & Sons, Inc., Hoboken New Jersey, 2nd edition, 2012. [p222]

A. D. Jassby and J. E. Cloern. *wq: Exploring water quality monitoring data*, 2014. URL http://CRAN.R-project.org/package=wq. R package version 0.4-1. [p225, 226]

D. Kahle and H. Wickham. ggmap: Spatial visualization with ggplot2. *The R Journal*, 5(1):144–161, 2013. URL http://journal.r-project.org/archive/2013-1/kahle-wickham.pdf. [p225, 229]

W. M. Kemp and J. M. Testa. Metabolic balance between ecosystem production and consumption. In E. Wolanski and D. S. McLusky, editors, *Treatise on Estuarine and Coastal Science*, pages 83–118. Academic Press, New York, 2012. [p227]

M. Kendall and A. Stuart. *The Advanced Theory of Statistics*, volume 3. MacMillan Publishing Company, New York, New York, 1983. [p224, 225]

A. McLeod, N. M. Mohammad, J. Veenstra, and A. El-Shaarawi. *cents: Censored time series*, 2014. URL http://CRAN.R-project.org/package=cents. R package version 0.1-41. [p222]

J. A. Needoba, T. D. Peterson, and K. S. Johnson. Method for the quantification of aquatic primary production and net ecosystem metabolism using in situ dissolved oxygen sensors. In S. M. Tiquia-Arashiro, editor, *Molecular Biological Technologies for Ocean Sensing*, pages 73–101. Springer, New York, 2012. [p227]

H. T. Odum. Primary production in flowing waters. *Limnology and Oceanography*, 1(2):102–117, 1956. [p225, 226]

K. S. Ro and P. G. Hunt. A new unified equation for wind-driven surficial oxygen transfer into stationary water bodies. *Transactions of the American Society of Agricultural and Biological Engineers*, 49 (5):1615–1622, 2006. [p227]

System-Wide Monitoring Program Data Analysis Training. SWMP data analysis training workshop provided at the 2014 NERRS/NERRA annual meeting, November 17, 2014. http://copepod.org/nerrs-swmp-workshop/. [p219]

J. Thébault, T. S. Schraga, J. E. Cloern, and E. G. Dunlavey. Primary production and carrying capacity of former salt ponds after reconnection to San Francisco Bay. *Wetlands*, 28(3):841–851, 2008. [p227]

Marcus W Beck
US Environmental Protection Agency
National Health and Environmental Effects Research Laboratory
Gulf Ecology Division
1 Sabine Island Drive, Gulf Breeze, FL 32651
USA
beck.marcus@epa.gov

CryptRndTest: An R Package for Testing the Cryptographic Randomness

by Haydar Demirhan and Nihan Bitirim

Abstract In this article, we introduce the R package **CryptRndTest** that performs eight statistical randomness tests on cryptographic random number sequences. The purpose of the package is to provide software implementing recently proposed cryptographic randomness tests utilizing goodness-of-fit tests superior to the usual chi-square test in terms of statistical performance. Most of the tests included in package **CryptRndTest** are not available in other software packages such as the R package **RDieHarder** or the C library TestU01. Chi-square, Anderson-Darling, Kolmogorov-Smirnov, and Jarque-Bera goodness-of-fit procedures are provided along with cryptographic randomness tests. **CryptRndTest** utilizes multiple precision floating numbers for sequences longer than 64-bit based on the package **Rmpfr**. By this way, included tests are applied precisely for higher bit-lengths. In addition **CryptRndTest** provides a user friendly interface to these cryptographic randomness tests. As an illustrative application, **CryptRndTest** is used to test available random number generators in R.

Introduction

Cryptographic random numbers constitute the heart of ciphering processes. Security of the transmitted information is mostly based on the quality of random numbers used to cipher the information. Due to efficiency considerations, pseudo random numbers that ensure some hard-to-achieve properties are used for ciphering in practice. There are a considerable amount of pseudo random number generators (RNG's) in the literature of cryptography. Suitability of these RNG's for use in cryptographic applications is evaluated based on statistical randomness tests that are specifically designed to test randomness at the level required for ciphering processes.

In a cryptographic randomness test, first, the empirical distribution of a test statistic is obtained over a random number sequence by various data manipulations. Then, a statistical goodness-of-fit test is applied to evaluate significance of the difference between the empirical distribution and its theoretical counterpart at a predetermined level of significance. The need for a certain level of randomness to ensure unpredictability in the cryptography context makes procedures used to check cryptographic and classical randomness different from each other. The manipulations of random number sequences are required to make the cryptographic randomness tests more sensitive to small deviations from the exact randomness than their classical counterparts. The null hypothesis of the test is "H_0 : Sequences generated by the RNG of interest are random." There are more than a hundred alternative tests for the evaluation of cryptographic randomness available (L'Ecuyer and Hellekalek, 1998).

In the literature, some of these tests are grouped into test batteries or test suites (L'Ecuyer and Simard, 2007; Marsaglia and Tsang, 2002). A detailed review of test batteries is given by Demirhan and Bitirim (2016). To be qualified as suitable, an RNG should be identified as random in a predetermined portion or all of the tests in a test battery. The basic test battery is introduced by Knuth (1998, 1981, 1969). Then, Marsaglia (1996) introduced the Diehard test battery composed of 12 randomness tests. Disadvantages of the Diehard test battery were overcame by another test battery called Dieharder that is introduced by Brown et al. (2014). Dieharder includes 26 cryptographic randomness tests. It is an improvement of the Diehard battery, provides a user friendly interface and a useful open source tool set for users of random numbers (Brown et al., 2014). The Dieharder test battery is implemented in the R package **RDieHarder** prepared by Eddelbuettel and Brown (2014). At the time of writing, Windows and OS X binaries are not available for this package. The US National Institute of Standards and Technology developed the NIST battery composed of 16 tests (Sýs et al., 2014; Sýs and Říha, 2014; Rukhin et al., 2010; Rukhin, 2001; Soto, 1999). The NIST battery is still used as a straightforward tool for formal certifications and accepted as a standard test battery. Sadique et al. (2012) reviewed the tests included in the NIST test battery. A suite of test batteries, TestU01, was introduced by L'Ecuyer and Simard (2007, 2014). TestU01 is a C library that combines most of the available randomness tests and RNGs in six test batteries (McCullough, 2006; L'Ecuyer and Simard, 2007). There are also smaller scale test batteries in terms of extensiveness. ENT was proposed by Walker (2014) that contains 5 statistics and tests. The Helsinki test battery is based on the Ising model and random walks on lattices and was proposed by Vattulainen et al. (1995). The Crypt-X test battery, which includes 6 tests, was developed by the Information Security Research Center at Queensland University of Technology (Sýs and Říha, 2014; Soto, 1999). The SPRNG test battery includes some tests from the battery of Knuth (Mascagni and Srinivasan, 2000). Ruetti (2004) combined Knuth, Helsinki, Diehard, and SPRNG batteries and proposed a test battery consisting of 37 statistical and physical tests.

In addition to the tests included in test batteries, there also exist recently proposed cryptographic randomness tests that are not performed by test batteries. Maurer (1992) proposed a statistical test for random bit generators. Hernandez et al. (2004) proposed a new test called Strict Avalanche Criterion (SAC). Ryabko et al. (2004) proposed an adaptation of the well-known chi-square test. This test is more efficient than the usual chi-square test in small samples. "Book Stack" and "Order" tests were proposed by Ryabko and Monarev (2005) for testing binary random bit sequences. Doganaksoy et al. (2006) proposed three randomness tests based on the random walk process. An advantage of these tests is that it is possible to calculate exact probabilities corresponding to the test statistics. The "Topological Binary Test" was introduced by Alcover et al. (2013) to test randomness in bit sequences. It counts different bit patterns of pre-determined length in a sequence of random bits.

Availability of a software implementing a test battery or even that of an individual cryptographic randomness test is a critical issue for the usefulness of the related test or battery. The library TestU01 is developed in ANSI C; hence, it is compiled by GNU tools instead of today's C compilers. Although TestU01 performs a wide variety of tests and their combinations, it lacks flexibility of implementation. Because the battery Dieharder is implemented in an R package, namely **RDieHarder**, it is more applicable and user-friendly than TestU01. However, unavailability of Windows and OS X binaries can be seen as a disadvantage that decreases its accessibility. A package for the implementation of the NIST battery is prepared for SUN workstations using ANSI C (Rukhin et al., 2010). Rukhin et al. (2010) provides a user guide for setting up the package and running the included tests. Ease of implementation of the NIST battery is similar with TestU01. For the implementation of individual randomness tests, there are also numerous R packages such as **randtests** (Caeiro and Mateus, 2014) or **DescTools** (Signorell, 2015). Although some of the tests included in these packages are also used to evaluate cryptographic randomness, they cover neither recently proposed tests nor those developed specifically to test cryptographic randomness.

The usual chi-square test is applied with nearly all of the cryptographic randomness tests in the literature. The mentioned implementations including those covered by R automatically apply the chi-square test. However, there are numerous alternatives to the chi-square goodness-of-fit test such as the Kolmogorov-Smirnov, Anderson-Darling, or Jarque-Bera tests. It is apparent that because statistical qualities of these tests are better than the chi-square test, there will be a gain in performance of cryptographic randomness tests when applied with better goodness-of-fit tests. Thus, we need software that is capable of conducting actual cryptographic randomness tests such as topological binary, book stack, etc. with goodness-of-fit tests better than the usual chi-square test in statistical performance. When the range and variety of cryptographic randomness tests implemented by software and practicability of available software are considered, this software should effectively implement new tests with various goodness-of-fit tests and has a user-friendly interface. The package **CryptRndTest** (Demirhan, 2016) contributes to satisfy this need.

The aim of this article is to describe and illustrate the use of the R package **CryptRndTest** (currently in version 1.2.2) that performs some of recently proposed and basic cryptographic randomness tests. The package includes the functions `adaptive.chi.square`, `birthday.spacings`, `book.stack`, `GCD.test`, `topological.binary`, and `random.walk.tests` to perform adaptive chi-square, birthday spacings, book stack, greatest common divisor, topological binary tests, and three tests based on the random walk process, respectively. To the best of our knowledge, the adaptive chi-square, topological binary, and the tests based on the random walk process are first implemented in software with package **CryptRndTest**. In addition to the chi-square procedure, these functions apply Anderson-Darling, Kolmogorov-Smirnov, and Jarque-Bera procedures when suitable. Because statistical performances of goodness-of-fit tests differ under various conditions, the application of different goodness-of-fit procedures is a beneficial feature. This is another important feature of package **CryptRndTest**. In addition, it has the following auxiliary functions: `GCD`, `GCD.q`, `GCD.big`, `Strlng2`, `toBaseTwo`, `toBaseTen`, and `TBT.criticalValue` to compute the greatest common divisor under different conditions of inputs, approximately calculate the Stirling number of the second kind when the inputs are large, make base conversions precisely with large inputs, and calculate critical values for the topological binary test.

The paper is organized as follows: in the next section, methodologies of the tests included in package **CryptRndTest** are briefly given. Details of algorithms used to manipulate integer and bit sequences are mentioned, and applications of goodness-of-fit procedures performed by package **CryptRndTest** are clarified. Parameter settings and limitations for each test are mentioned. Finally, as an illustrative application of package **CryptRndTest**, random number generators available in R are tested by using the proposed package under different sequence and bit-length conditions. By this application, implementation performance of the package is analyzed, recently proposed tests are evaluated, and usage of package **CryptRndTest** is illustrated.

Performed tests

Adaptive chi-square

The adaptive chi-square test was introduced by Ryabko et al. (2004). It is empirically demonstrated by Ryabko et al. (2004) that the adaptive chi-square test is more efficient than the classical chi-square test in the identification of non-random patterns in samples smaller than those required by the chi-square test. For example, when we work with 64-bit numbers the length of the alphabet is 2^{64}; hence, we need to have a sequence of length greater than $5 \cdot 2^{64}$ to apply the classical chi-square test safely. The logic behind the test is to divide the alphabet into subsets and perform the chi-square test over subsets instead of individual elements of the sample. By this way, subsets are considered as a new alphabet and a new null hypothesis and its alternative are formed over the subsets. Because the number of categories required to test new hypotheses is equal to the number of subsets, the chi-square test is applied with much smaller samples. To conclude randomness, it is expected to observe a uniformity in the distribution of input numbers into the subsets. Deviations from this uniformity are detected by the adaptive chi-square test.

The function `adaptive.chi.square()` is called to apply the test. It implements the following pseudo-code algorithm:

Algorithm 1.

1. Input data as a matrix of bits or a vector of integers, the number of subsets (S) that the alphabet will be divided into, and proportion of training data set;
2. If data is represented by bits, transform data to base-10;
3. Divide whole data set into training and testing subsets with regarding input weights;
4. Identify the numbers that are seen in the sequence of interest at least once;
5. Find the frequency of occurrences for each element of the alphabet in training and testing subsets;
6. For $i = 1, \ldots, S$, find the frequency of elements that are seen i-times in the training and testing subsets;
7. Apply the two-sample chi-square test with the expected and observed counts obtained at the previous step over the training and testing subsets, respectively;
8. Return value of the test statistic, corresponding p-value, and the decision on the null hypothesis.

While working with integers, the alphabet corresponds to the range of considered numbers. For instance, if 32-bit numbers are being tested, the alphabet in Algorithm 1 includes the numbers between 0 and $2^{32} - 1$. At step 4, we do not form the whole alphabet, instead we count the numbers (words) that are seen at least once; and hence, the rest of the numbers have zero count. At step 7, the degrees of freedom of the test is $S - 1$.

Parameters of the adaptive chi-square test are: weight of training and testing samples (r), the length of the considered number sequence (n), and the number of subsets (S) that the alphabet is divided into. Ryabko et al. (2004) do not give strict rules for the determination of values of these parameters. They suggest to run some experiments to find the values of parameters that provide the highest statistical performance such as power and specificity. Because such a study would not be cost-effective for an individual application of the test, at least, the user may evaluate sensitivity of test results to the values of S and r. In the function `adaptive.chi.square()`, we set $r = 0.5$ by default. The value of S is set by the user. That of n is determined by the length of input data. Because input data is a random sample from the RNG of interest, the value of n should be increased with increasing bit-length to successfully represent the range of numbers that will be generated by the RNG. When bit-length is greater than 64, we utilize the package **Rmpfr** (Maechler, 2015) to work with higher precision.

Algorithm complexity of the function `adaptive.chi.square()` is $O(n^2)$ in the worst case. Required memory is directly related to the length of the input sequence. Due to the algorithm complexity of the function used to identify unique numbers at step 4, implementation time of the function `adaptive.chi.square` increases quadratically along with the length of the input sequence.

Birthday spacings

The Birthday Spacings test was given by Marsaglia and Tsang (2002). It focuses on the number of duplicated values of spacings between ordered birthdays among a year of pre-determined length. The observed duplication patterns in input numbers are compared with the patterns that should be observed under randomness. Thus, the birthday spacings test detects deviations from randomness

by focusing on repetition frequency of numbers to ensure uniformity. Marsaglia and Tsang (2002) propose that the number of duplicated values is approximately distributed according to the Poisson distribution. They also derive an expression for the mean rate of the Poisson distribution.

The function `birthday.spacings()` is employed to run the test. It implements the following pseudo-code algorithm:

Algorithm 2.

1. Input data as a vector of integers of size n, the number of birthdays (m), the length of year (N), the mean rate of the theoretical Poisson distribution (λ), and the number of classes (k) that is constructed for goodness-of-fit tests;
2. Reshape the first $m \cdot \lfloor n/m \rfloor$ elements of input vector as a matrix of $\lfloor n/m \rfloor$ rows and m columns;
3. Sort each row of the matrix of step 2 according to the values in columns;
4. For each row, find the distance between columns of the sorted matrix by extracting the values in the columns at the previous step;
5. Count duplicated values among the distances obtained at step 4;
6. Calculate class probabilities over the Poisson distribution with mean rate λ for $x = 0, \ldots, k$, and assign the rest of probability mass to the $(k+1)$-th class;
7. Calculate expected frequencies corresponding to the probabilities obtained at the previous step;
8. Replicate the expected counts to form the corresponding sample;
9. Apply the Anderson-Darling test to compare goodness-of-fit of the samples obtained at steps 5 and 8;
10. Apply the Kolmogorov-Smirnov test to compare goodness-of-fit of the samples obtained at steps 5 and 8;
11. Construct frequency table of the counts obtained at step 5;
12. Apply chi-square test over the frequency tables obtained at steps 7 and 11;
13. Return the values of test statistics, corresponding p-values, and decisions on the null hypothesis.

At step 2 of Algorithm 2, each row of the reshaped matrix includes birthdays in columns. Total number of rows determines the size of the sample that is used in goodness-of-fit tests applied at steps 9, 10, and 12. Manipulation of the input vector according to the birthday spacings test is completed at step 5. This manipulation produces the empirical sample in testing the goodness-of-fit to the Poisson distribution. The Anderson-Darling test at step 9 is applied by using function `ad.test` from the package **kSamples** (Scholz and Zhu, 2016). The Kolmogorov-Smirnov test at step 10 is applied by using function `ks.test` from the package **stats**.

Marsaglia and Tsang (2002) give some insight into the optimal values of parameters. The mean rate is $\lambda = m^3/(4n)$. They state that for an RNG, it is harder to pass this test for increasing values of either m or n. Specifically, the case with $m = 4096$ and $n = 2^{32}$ is qualified as a compelling setting for 32-bit generators. Length of the input sequence is another important parameter. Because the size of the sample used in testing the goodness-of-fit is equal to $\lfloor n/m \rfloor$, the length of the input sequence (n) should be chosen large enough to apply the goodness-of-fit tests appropriately.

Algorithm complexity of the function `birthday.spacings()` is $O(n^2)$ in the worst case. The limitation of `birthday.spacings()` is directly related with the value of m. For all combinations of m and n suggested by Marsaglia and Tsang (2002), λ is equal to 4. Following this logic, when $n = 2^{64}$ the value of m giving $\lambda = 4$ is 6,658,043. In this case, for a reliable application of goodness-of-fit tests at steps 9, 10, and 12, we need at least 133,160,860 integers and correspondingly 8,522,295,040 bits. For bit lengths higher than 32, the value of λ can be taken as 2. For instance, when $n = 2^{64}$, the corresponding value of m is 5,284,492. Thus, decreasing the value of λ does not overcome the need for a huge data set for a reliable testing. Note that use of huge data set for testing is a memory consuming operation.

Book stack

The Book Stack test was proposed by Ryabko and Monarev (2005). Positions of the numbers on a stack are taken into consideration. In this test, randomness implies that frequency of finding each number at each position is equally likely. Departures from this equality mean that some of the words are seen more frequently in contrast to the nature of randomness. The book stack test focuses on non-uniform patterns and frequent repetitions of input numbers to detect deviations from randomness by means of unexpected autocorrelation patterns and non-uniformity.

The function `book.stack()` implements the following pseudo-code algorithm to run the test:

Algorithm 3.

1. Input data as a matrix of bits or a vector of integers and the number of subsets (k) that the alphabet will be divided into;
2. If data are represented by bits, transform data to base-10;
3. Form an array that includes the numbers from 1 to the number of unique words in the input sequence;
4. Write each element of the input vector in place of the first element of the array formed at the previous step, and move the other elements except the one written to the first cell of the array one step right;
5. Record the array obtained at the previous step;
6. Go back to step 4 until all elements of the input vector are taken into account;
7. Divide the whole alphabet into k non-overlapping subsets $(A_1, A_2, \ldots, A_k)$;
8. For each subset of the alphabet, find the frequency of occurrences of the number corresponding to the position of each element of input vector in the arrays formed at steps 4 and 5;
9. Apply the chi-square test with expected counts equal to $n \cdot A_i$, where $i = 1, \ldots, k$ and n is the length of input vector or number of columns of input matrix;
10. Return the value of test statistic, corresponding p-value, and decision on the null hypothesis.

In order to get an integer number of subsets, the length of input vector should be determined to get an integer as the length of subsets. Optimal value for the length of input vector is given as $n \approx B \cdot 2^{B/2}$, where B is the bit-length of the considered RNG (Ryabko and Monarev, 2005; Doroshenko and Ryabko, 2006; Doroshenko et al., 2006). For an appropriate determination of number of subsets, k, Ryabko and Monarev (2005) suggest performing an empirical study. As for an appropriate bit-length, it is mentioned by Ryabko and Monarev (2005) that it is hard to set up a sensible test with much higher bit-lengths.

Algorithm complexity of the function `book.stack()` is $O(n^2)$ in the worst case. The limitation of the Book Stack test is based on the bit-length of the considered RNG. For example, for $B = 64$ the length of the input vector is calculated as $1.37 \cdot 10^{11}$ and we need 1 terabyte memory whereas the memory requirement is 4 megabytes for $B = 32$. Due to both memory and sensibility issues, it is not appropriate to work with high bit-lengths such as 64.

Greatest common divisor

Two tests proposed by Marsaglia and Tsang (2002) are based on the number of required iterations (k) and the value of the greatest common divisor (GCD) obtained in the GCD operation. When perceived as random variables, both k and GCD are independently and identically distributed and their distributions can be obtained under randomness. Marsaglia and Tsang (2002) derived distributions of k with an empirical study and that of GCD theoretically under the null hypothesis of randomness. Departures from randomness imply nonconformity between empirical and theoretical distributions of k and GCD. Thus, these tests focus on the deviations from independence and uniformity.

The function `gcd.test()` is called to apply the test. The following pseudo-code algorithm is implemented by `gcd.test()` when all of the goodness-of-fit tests are set to `TRUE`:

Algorithm 4.

1. Input data as an $N \times 2$ matrix of integers, mean and standard deviation of theoretical normal distribution of k;
2. Constitute a pair of numbers from each row of input matrix;
3. Apply the GCD operation to each pair formed at the previous step;
4. Store values of k for N pairs;
5. If the obtained GCD is less than 3, store it as 3 and if that of GCD is greater than 35, store it as 35;
6. Generate a random sample of size N from the normal distribution with input values of mean and standard deviation.
7. If the tests based on k will be conducted, go to the next step, otherwise go to step 13;
8. Apply the two sample Kolmogorov-Smirnov test in a two-sided setting to samples obtained at steps 4 and 6;
9. Apply the chi-square test to samples obtained at steps 4 and 6;

10. Standardize the values of k by using its empirical mean and standard deviation;
11. Apply the Jarque-Bera test to the standardized sample of step 10;
12. Apply the Anderson-Darling test to samples obtained at steps 4 and 6;
13. If the tests based on the GCD will be conducted, go to the next step, otherwise go to step 19;
14. Construct the cumulative distribution function (cdf) of the probability function (pf) of GCD given by Marsaglia and Tsang (2002);
15. Obtain theoretical frequencies for the GCD over the cdf of step 14. Specifically, if theoretical frequency of the GCD is less than 3, store it as 3 and if that of the GCD is greater than 35, store it as 35;
16. Replicate the expected counts to form the corresponding sample;
17. Apply the two sample Kolmogorov-Smirnov test in a two-sided setting to samples obtained at steps 5 and 16;
18. Apply the chi-square test to samples obtained at steps 5 and 16;
19. Return the values of calculated test statistics, corresponding p-values, and decisions on the null hypothesis.

Mean and standard deviation of the theoretical normal distribution for bit lengths other than 32 are not given by Marsaglia and Tsang (2002). We conducted extensive empirical studies, details of which are mentioned in the following sections, to obtain these parameters and tabulated obtained values in Table 3.

When bit-length is increased, corresponding value of GCD mostly becomes greater than 35; hence, the operation at step 15 of Algorithm 4 gets unreasonable. Thus, we observe that it is not appropriate to conduct tests based on the GCD for high bit-lengths such as 128.

The Kolmogorov-Smirnov and chi-square tests at steps 8 and 17, and 9 and 18 are applied by using functions `ks.test` and `chisq.test` from the package **stats**, respectively. The Jarque-Bera test at step 11 is implemented by using the function `jarque.bera.test` from the package **tseries** (Trapletti and Hornik, 2015). The Anderson-Darling test is applied by using the function `ad.test` from the package **kSamples**.

Calculations of the number of required iterations and the value of the GCD are time consuming tasks for bit-lengths greater than 64. To overcome this difficulty, we prepared three functions to calculate GCD-related variables. The first function `GCD.q` computes the number of required iterations, the value of the GCD, and the sequence of partial quotients by using the Euclidean algorithm. The function `GCD` is the recursive version of the Euclidean algorithm and it only provides the number of required iterations and the value of the GCD. The function `GCD.big` applies the Euclidean algorithm over multiple precision floating point numbers using package **Rmpfr** and provides all three outputs related with the GCD operation. While `GCD` is the fastest one, `GCD.big` gives the most precise results. It is also possible to use the binary GCD algorithm to decrease the implementation time. However, in this case it is not possible to apply tests over the number of required iterations of the Euclidean algorithm. When the GCD operation is done recursively, the algorithm complexity of `gcd.test()` is $O(\log(a))$, where a is the maximum initial input to the recursive algorithm. Memory requirement for GCD tests is directly related with the value of N.

Random walk tests

In the literature, binary sequences are analyzed in detail by using the random walk process. Doganaksoy et al. (2006) proposed three tests based on the random walk stochastic process. In a random walk process, magnitude or direction of each change is determined by chance; hence, a random walk is random if increment and decrement probabilities are equal to each other. Therefore, random walk processes provide a good basis for randomness. In a random walk, a part of the sequence that intersects the x-axis with two successive points is called excursion, and over all excursions, the maximum distance from the x-axis is defined as height, and the vertical distance between minimum and maximum points over the y-axis is called expansion. Thus, we have three characteristics of the random walk process to observe deviations from randomness. The corresponding tests are called Random Walk Excursion, Random Walk Height, and Random Walk Expansion. If there is a trend in the process, the input sequence fails in the excursion test. The height test focuses on the moves with very low or high magnitude to detect non-randomness. The expansion test focuses on the anomalies in amplitude of the walk to identify non-random patterns. Because the exact probabilities corresponding to test statistics are calculated, the tests proposed by Doganaksoy et al. (2006) are also applicable for small sample sizes.

The function `random.walk.tests()` is called to apply three tests, selectively. The following pseudo-code algorithm is implemented by `random.walk.tests()` when all of the tests are to be applied:

Algorithm 5.

1. Input data as a matrix of bits of dimension $B \times k$, where B is the bit length and k is the length of input sequence;
2. Transform the input values from $\{0, 1\}$ to $\{-1, 1\}$;
3. To apply the expansion, excursion, and height tests go to steps 4, 6, and 7, respectively;
4. For each non-overlapping set of length B, sum adjacent bits starting from the first bit and increasing by one at each iteration (By this way, we get B summations for each number of interest);
5. For the Expansion test, count and store the summations of the previous step equal to zero;
6. For the Excursion test, calculate the maximum summation and the absolute value of the minimum summation among those of step 4 and store their sum;
7. For the Height test, store the absolute maximum of summations obtained at step 4;
8. Calculate theoretical cdf's and pf's for the tests regarding bit-lengths and probabilities tabulated by Doganaksoy et al. (2006);
9. Calculate empirical cdf's and pf's over the counts obtained at steps 5, 6, and 7;
10. Replicate the expected and empirical pf's to form the corresponding samples;
11. Apply the Anderson-Darling test to samples obtained at the previous step;
12. Apply the two sample Kolmogorov-Smirnov test in a two-sided setting to samples obtained at step 10;
13. Apply the chi-square test to samples obtained at step 10;
14. Return the values of calculated test statistics, corresponding p-values, and decisions on the null hypothesis.

The Anderson-Darling test at step 9 is applied by using function `ad.test` from the package **kSamples**. The Kolmogorov-Smirnov test at step 10 is applied by using function `ks.test` from the package **stats**. The chi-square test at step 11 is the classical application of the test without using a predefined function. If one of the tests is not applied, all the results related with that test in output are set to -1.

Algorithm complexities of expansion, excursion, and height tests are $O(B)$, $O(B\lfloor k \cdot B \rfloor)$, and $O(B\lfloor k \cdot B \rfloor)$, respectively. The limitation of the tests is unavailability of theoretical cdf's for bit-lengths other than 32, 64, 128, and 256. Therefore, using the information given by Doganaksoy et al. (2006) the excursion is applied for bit-lengths of 16, 32, 64, 128, and 256; the height test is applied for bit-lengths of 64, 128, 256, 512, and 1024; and the expansion test is applied for bit-lengths of 32, 64, and 128. Although the size of required memory increases along with the length of the input sequence, it is possible to apply the tests with reasonable sequence lengths without causing memory pressure.

Topological binary

The topological binary test was proposed by Alcover et al. (2013) to test the randomness in bit sequences. The logic behind the test is based on the number of different fixed-length bit patterns in a bit sequence. Frequency of distinct non-overlapping bit patterns over the sequence of interest is influential on the test result. In case of randomness, we expect to have many different bit patterns in the input sequence. The main strength of the topological binary test is that it focuses on the number of bit patterns rather than frequency of occurrence of numbers. Because the exact distribution of the test statistic is derived, it is possible to apply the test for short bit sequences.

The function `topological.binary()` implements the following pseudo-code algorithm to run the test:

Algorithm 6.

1. Input data as a $B \times k$ matrix of bits, where B is the bit-length and k is the length of considered number sequence, and the critical value;
2. Find and store non-overlapping blocks of length B;
3. Count the number of different B-bit patterns that appear across all the k blocks;

	Function	Call
Test	GCD.test()	GCD.test(x,KS = TRUE,CSQ = TRUE,AD = TRUE,JB = TRUE, test.k = TRUE, test.g = TRUE, mu, sd, alpha = 0.05)
	random.walk.tests()	random.walk.tests(x,B = 64,Excursion = TRUE, Expansion = TRUE, Height = TRUE, alpha = 0.05)
	birthday.spacings()	birthday.spacings(x,m = 128,n = 2^{16},alpha = 0.05,lambda, num.class = 10)
	adaptive.chi.square()	adaptive.chi.square(x,B,S,alpha = 0.05,bit = FALSE)
	book.stack()	book.stack(A,B,k = 2,alpha = 0.05,bit = FALSE)
	topological.binary()	topological.binary(x,B,alpha = 0.05,critical.value)
Auxiliary	Strlng2()	Strlng2(n,k,log = TRUE)
	GCD()	GCD(x,y)
	GCD.q()	GCD.q(x,y)
	GCD.big()	GCD.big(x,y,B)
	TBT.CriticalValue()	TBT.criticalValue(m,k,alpha = 0.01,cdf = FALSE,exact = TRUE)
	toBaseTen()	toBaseTen(x,m = 128,prec = 256,toFile = FALSE,file)
	toBaseTwo()	toBaseTwo(x,m = 128,prec = 512,num.CPU = 4)

Table 1: Usage of test and auxiliary functions of package **CryptRndTest**.

4. If the result of step 3 is less than one, then reject the null hypothesis;
5. Else if the result of step 3 is greater than $\min(k, 2^B)$, then do not reject the null hypothesis;
6. Else if the result of step 3 is less than the input critical value, then reject the null hypothesis;
7. Else do not reject the null hypothesis;
8. Return the result of step 3 as the value of test statistic and the decision on the null hypothesis.

Although the exact distribution of test statistic is derived by Alcover et al. (2013), calculation of the Stirling numbers of the second kind with large inputs is required with bit-lengths greater than 16 for the calculation of the cdf of the test statistics. Therefore, it is hard to obtain the critical value of the test for large bit-lengths by using available functions in R packages such as the function `Stirling2` of package **copula** (Hofert et al., 2015). This case is a limitation of the function `topological.binary()`. To overcome this limitation of the test, we prepared the function `TBT.CriticalValue` to calculate required critical values for testing. Algorithm complexity of the function `topological.binary()` is $O(n^2)$ in the worst case. The required memory to run the topological binary test is related with the value of k.

Auxiliary functions

The package **CryptRndTest** has seven auxiliary functions, i.e., `Strlng2()`, `GCD()`, `GCD.q()`, `GCD.big()`, `toBaseTwo()`, `toBaseTen()`, and `TBT.CriticalValue()`. These functions are also suitable for individual use. `Strlng2()` is used to calculate critical values for the topological binary test implemented by `TBT.CriticalValue()`. `GCD()` and `GCD.q()` are called to calculate the greatest common divisor in the GCD test implemented by `gcd.test()`. Three possible outcomes of the greatest common divisor operation are the number of iterations, the sequence of partial quotients, and the value of greatest common divisor. `GCD()` provides all of these outcomes for any pair of integers excluding zero. Functions `toBaseTwo()` and `toBaseTen()` are used for base conversion from base 2 to 10 and vice versa for large integers.

The function `Strlng2()` is used to compute the natural logarithm of Stirling numbers of the second kind for large values of inputs in an approximate manner by the approaches of Bleick and Wang (1974) and Temme (1993). In this approach, Lambert W functions are employed at the log scale to overcome memory overflows.

Due to the large factorials in the calculation of Stirling numbers of the second kind, it is nearly impossible to compute the exact cdf of the topological binary test statistic for higher bit lengths without memory flows in R. The function `TBT.CriticalValue()` implements an approach for the calculation of the cdf and approximately computes the required critical value for the topological binary test at a given level of α. Because `TBT.CriticalValue()` utilizes `Strlng2()`, accuracy of results decreases with increasing bit lengths and number of words under consideration. It is also possible to make exact calculations by `TBT.CriticalValue()`. In this case, the function `Stirling2` from the package **gmp** (Lucas et al., 2014) is employed instead of `Strlng2()`. Because package **gmp** uses multiple precision arithmetic, implementation time of `TBT.CriticalValue()` considerable increases. User should evaluate the trade off between implementation time and high precision.

Arguments of main and auxiliary functions of package **CryptRndTest** are summarized in Table 1.

	Sequence length		
Bit	Short (I)	Medium (II)	Long (III)
8	256	32768	65536
16	16384	65536	131072
32	32768	131072	262144
64	131072	262144	524288
128	131072	262144	524288

Table 2: Lengths of random number sequences for different patterns.

A numerical illustration

As a numerical illustration of the package, we employed package **CryptRndTest** to test the randomness of RNG's available in R. By this way, we aim to get results of the tests that have not been applied to RNG's of interest yet, figure out implementation performance of package **CryptRndTest** under various scenarios, and illustrate some issues on the determination of parameters of the tests for considered scenarios. Note that it is impossible to observe the ability to control type-I error (rejection of randomness hypothesis while it is actually true) for the tests with an empirical study such as conducted in this section. Additionally, a more thorough investigation would be necessary to be able to reliably assess the algorithms, but this is out of scope of this article.

RNG's of interest are Wichmann-Hill (WH), Marsaglia-Multicarry (MM), Super-Duper (SD), Mersenne-Twister (MT), Knuth-TAOCP-2002 (KT02), Knuth-TAOCP (KT), and L'Ecuyer-CMRG (LE) (see the function `Random` in the **base** package for the details of these RNG's). Applied tests are topological binary (TBT), adaptive chi-square (Achi), birthday spacings (BDS), random walk expansion (RWT.Exp), random walk height (RWT.Hei), random walk excursion (RWT.Exc), book stack (BS), and greatest common divisor (GCD). TBT, RWT.Exp, RWT.Hei, and RWT.Exc tests work with binary numbers while the rest of the tests take integers as input. BDS and RWT tests are applied separately with each of Anderson-Darling, Kolmogorov-Smirnov, and chi-square goodness-of-fit tests, and the GCD test is applied separately with each of Anderson-Darling, Kolmogorov-Smirnov, Jargue-Bera, and chi-square goodness-of-fit tests. The total number of applied randomness tests is 21. All the tests are applied at both 0.01 and 0.05 levels of significance and 8, 16, 32, 64, and 128-bit lengths. Considered lengths of random number sequences for each bit-length are given in Table 2.

Because we get unreasonable implementation times for longer sequences at the level of 128-bit, the same sequence lengths as 64-bit are considered for 128-bit numbers.

To conduct the adaptive chi-square test, we need to determine the value of argument `S` and the proportions of training and testing samples. The latter one is taken equal. As for the value of `S`, we did not detect a significant change in the test results observed for medium sequence length for all bit-lengths for $S = 2, 3, 4$ in pilot runs. The values greater than 4 increase the implementation time whereas small values decrease resolution. Thus, it is taken as 4 for all bit-lengths to work with a reasonable degrees of freedom in the chi-square test. Also, the adaptive chi-square test is applied for all bit-lengths.

Arguments of the birthday spacings test are the number of birthdays (`m`), the length of year (`n`), the mean rate of the theoretical Poisson distribution (`lambda`), and the number of classes (`num.class`), which is used for goodness-of-fit tests. In the experiments, the argument `m` was taken as 8, 128, and 4096 for 8, 16, and 32-bit-lengths, respectively. The argument `n` was set to 2^B, where B is the bit-length. The argument `lambda` was calculated by the formula given by Marsaglia and Tsang (2002). The argument `num.class` was set to 5 and 10 for 8 and 16-bit and higher lengths, respectively.

For the book stack test, length of the sample (`n`) should be determined and data should be prepared according to the value of `n`. Also, the number of subsets that the alphabet will be divided into (`k`) should be determined. The formula proposed by Ryabko and Monarev (2005) is used to calculate the value of `n`, and we set `k=n/`B.

In the GCD test procedure, tests are conducted for two outputs of the GCD operation, i.e., the number of iterations required to find GCD (k) and GCD (g) itself. The population distribution of k is well approximated by a normal distribution and parameters of the normal distribution are given by Marsaglia and Tsang (2002) for 32-bit integers after an extensive numerical study. We observed that the parameters of the population distribution differ for different bit-lengths and conducted a numerical study to figure out the values of parameters for considered bit-lengths. For this study, 10^6 30-bit true random numbers were obtained from the web service "www.random.org." Then, they were converted to 8, 16, 32, 64, and 128-bit numbers. The GCD operation was applied and mean (`mu.GCD`) and standard deviation (`sd.GCD`) of k were obtained as given in Table 3 after checking the normality of the empirical distribution by means of descriptive statistics and the Anderson-Darling goodness-of-fit

Bit	mu.GCD	sd.GCD
8	3.9991	1.6242
16	8.8784	2.3664
32	18.4023	3.4000
64	31.3269	4.3349
128	31.8390	4.3678

Table 3: Mean and standard deviation of population distribution of k.

	$\alpha = 0.01$			$\alpha = 0.05$		
Bit	Short	Medium	Long	Short	Medium	Long
8	153	NA	NA	153	NA	NA
16	14423	41268	NA	14423	41266	NA
32	32767	131066	262129	32767	131066	262129
64	131070	262113	523264	131070	262113	524264
128	131072	262144	524288	131072	262144	524288

NA: not available.

Table 4: Critical values for topological binary test.

test. The values obtained for 32-bit are very close to those obtained by Marsaglia and Tsang (2002).

As expected, the mean of k increases along with bit-length, and it approaches 35 (Marsaglia and Tsang, 2002). The mild increase in the values of standard deviations is due to the increasing range of the numbers that can be generated with a given bit-length. Also, the GCD test is applied for all bit-lengths. However, nearly for all 128-bit random numbers, $g > 35$. Due to the operation done at step 15 of Algorithm 4, it is unreasonable to conduct the GCD test over g for 128-bit numbers.

The topological binary test is also applied for all bit-lengths. Critical values for the topological binary test are calculated by using the function `TBT.criticalValue()` for each bit and sequence length combination and presented in Table 4. Because the length of sequence being tested cannot be longer than $2^m - 1$, where m is the bit-length, critical values for medium and long sequences at 8-bit and for long sequences at 16-bit levels are not available in Table 4.

In the application, random numbers were generated by using the same seed `283158301`. Let `sim.data` be an integer vector including data to be tested. It is reshaped with the following code according to bit-length `B`:

```
if (B <= 64) {
  sim.data <- matrix(data = sim.data, ncol = len, byrow = FALSE)
} else {
  sim.data <- mpfrArray(sim.data, prec = B)
}
```

The adaptive chi-square, random walk, and topological binary tests were straightforwardly called with the mentioned arguments. For the book stack test, the following code is employed:

```
n <- B * (2^(B/2))
dat.BS <- sim.data[1:round(n/B)]
BS <- book.stack(x = dat.BS, B = B, k = n/B, alpha = 0.01, bit = FALSE)
print(BS)
```

For the GCD tests, the input number sequence was divided into two-sequences and the tests were applied with the following code:

```
if (len%%2 == 1) {
  len <- len - 1
}
len2 <- len/2
if (B <= 64) {
  dat.new <- array(NA, dim = c(len2,2))
  dat.new[1:len2,1] <- sim.data[1:len2]
  dat.new[1:len2,2] <- sim.data[(len2+1):len]
} else {
  dat.new <- mpfrArray(NA, prec = m, dim = c(len2,2))
  dat.new[1:len2, 1] <- sim.data[1:len2]
  dat.new[1:len2, 2] <- sim.data[(len2+1):len]
```

		Tests							
Bit	Length	TBT	Achi	BDS	RWT.Exp	RWT.Hei	RWT.Exc	BS	GCD
8	32768	0.62	2.88	0.46	NA	NA	NA	< 0.01	1.31
16	65536	1.70	5.70	0.46	NA	NA	4.33	0.02	3.74
32	131072	6.68	10.88	2.10	NA	0.26	15.99	4253.01	12.32
64	262144	32.05	86.31	NA	84.21	88.74	64.68	NA	37.36
128	262144	77.16	10121.34	NA	221.16	196.96	149.29	NA	2657.62

TBT: topological binary, Achi: adaptive chi-square, BDS: birthday spacings, RWT: random walk, Exp: expansion, Hei: height, BS: book stack, GCD: greatest common divisor, Length: the length of random number sequence, NA: not available.

Table 5: Mean implementation time for each test in seconds.

```
}
EBOB <- GCD.test(x = dat.new, B = m, mu = mu.GCD, sd = sd.GCD, test.g = do.test.g)
print(EBOB)
```

where `len` is the length of the input sequence. Whole code snippets used to implement experiments are given in the vignette of the package **CrpytRndTest**.

Random number sequences used for the performance analysis are of medium length given in Table 2 and generated by the WH generator under each bit level. Five replications were made for each test. Mean implementation times calculated over five replications are shown in Table 5 in seconds. All variances of implementation times are less than 0.01. BDS, RWT, and BS tests were not applied at all bit-lengths due to reasons explained in the relevant sections.

Implementation times of all tests from 8 to 64-bit levels are all sufficient. For 128 bits, most of the implementation times of Achi and GCD tests are taken by finding unique values in a sequence composed of multiple precision floating-point (mpf) numbers at step 4 of Algorithm 1 and the value of gcd for mpf numbers at step 3 of Algorithm 4, respectively. For these operations, mpf numbers are used via the package **Rmpfr**. The package **Rmpfr** is based on the GMP GNU library and provides an interface from R to C (Maechler, 2011, 2015). Due to the use of mpf numbers via the package **Rmpfr**, there is a considerable increase in implementation time of Achi and GCD tests at 128-bit level. However, the gain in precision is worth the delay in implementation of these tests. Performances of the tests working with binary numbers are all sufficient at the 128-bit level. Implementation time of the BS test exponentially increases along with the bit-length. Although it is reasonable for 32 bits, application of the test for higher bit-lengths requires an unreasonable amount of time for implementation.

All the tests were applied at both 0.01 and 0.05 levels of significance. The null hypothesis is "H_0 : Sequences generated by the RNG of interest are random" for all tests. For both levels of significance, success rates of RNGs over the total number of applied tests are given in Table 6. Detailed test results for the 0.05 level of significance are presented in the vignette of the package **CrpytRndTest**. The total number of applied tests is given in the last row of Table 6 for each test scenario. For example, because the birthday spacings test is not applied for 64 bit-length, the total number of applied tests is 17 for all sequence lengths. Note that the values given in Table 6 should not be confused with issues related with statistical performance of the tests such as type I error or power. Table 6 represents the proportion of RNG's that did not fail in the given number of tests. In addition, because each test is applied individually, the information presented by Table 6 should not be perceived as the results of the application of a test battery.

In general, the proportion of success decreases with increasing sequence and bit-lengths. According to the proportions of success, performance of the WH generator is satisfactory for 16 and 32-bit numbers for all sequence lengths. The reason of getting a decreasing success rate with increasing bit-length is that the random walk tests with all goodness-of-fit tests and the GCD test with the Jarque-Bera goodness-of-fit test reject the randomness hypothesis while the rest of the tests mostly accept the hypothesis for bit-lengths greater than 32. In detail, the WH generator successfully passes both of the TBT and Achi tests nearly in all bit-sequence length combinations. Results of AD and KS goodness-of-fit tests applied under both BDS and GCD tests (with k) are similar, and the CS test more likely decides randomness of the WH generator. It is unsuccessful in passing the random walk tests for high bit-lengths. The BS test concludes WH's randomness under all of the test conditions. The GCD with the JB goodness-of-fit test rejects the null hypothesis of randomness under all test conditions but the first one. At the 0.01 level of significance, there is nearly no change in the results. The WH generator passes the GCD test with CS goodness-of-fit test for k at (8, I), (8, II) and (32, I) scenarios, and the BDS test with the AD goodness-of-fit test at (16, II).

According to the proportions of success, the SD generator mostly passes the tests for 16 and 32-bit integers for all sequence lengths, and 8-bit integers for short and long sequences. Detailed test results

Level of Significance	RNG	Bit-length: 8			16			32			64			128		
		Sequence length: I	II	III	I	II	III	I	II	III	I	II	III	I	II	III
0.01	WH	0.92	0.58	0.50	0.93	1.00	0.86	0.86	0.93	0.93	0.47	0.47	0.47	0.33	0.27	0.33
	SD	0.92	0.58	0.75	0.93	0.93	0.93	0.93	0.93	0.93	0.41	0.47	0.35	0.33	0.33	0.27
	MT	0.92	0.58	0.58	1.00	1.00	0.93	0.93	0.93	0.93	0.47	0.41	0.47	0.33	0.33	0.33
	MM	0.92	0.58	0.75	0.93	0.93	1.00	1.00	0.93	0.93	0.47	0.41	0.35	0.33	0.33	0.27
	LE	0.92	0.67	0.58	0.93	0.93	0.79	1.00	0.93	0.93	0.47	0.47	0.47	0.33	0.27	0.33
	KT	0.92	0.67	0.58	0.93	0.93	0.93	1.00	0.93	0.93	0.41	0.47	0.47	0.27	0.33	0.33
	KT02	0.92	0.67	0.58	1.00	0.93	1.00	1.00	0.93	0.86	0.47	0.41	0.47	0.27	0.33	0.33
0.05	WH	0.83	0.50	0.50	0.93	0.93	0.86	0.79	0.93	0.93	0.47	0.47	0.47	0.33	0.27	0.33
	SD	0.92	0.58	0.75	0.93	0.93	0.86	0.86	0.79	0.86	0.41	0.47	0.29	0.33	0.33	0.27
	MT	0.67	0.50	0.58	0.93	0.86	0.86	0.93	0.86	0.93	0.41	0.41	0.47	0.33	0.33	0.33
	MM	0.92	0.58	0.75	0.93	0.93	0.93	1.00	0.86	0.93	0.47	0.41	0.35	0.33	0.33	0.27
	LE	0.92	0.67	0.58	0.93	0.93	0.79	0.93	0.86	0.93	0.47	0.41	0.47	0.33	0.27	0.33
	KT	0.92	0.58	0.58	0.93	0.86	0.93	1.00	0.93	0.93	0.41	0.41	0.47	0.27	0.20	0.33
	KT02	0.83	0.58	0.42	1.00	0.93	0.93	1.00	0.93	0.79	0.47	0.41	0.47	0.27	0.33	0.33
Number of tests		12	12	12	15	15	15	15	15	15	17	17	17	15	15	15

Table 6: Success rates for RNGs over the tests applied by package **CryptRndTest**.

for the SD generator at the 0.05 level of significance are similar to that of the WH generator for the TBT, Achi, BDS, RWT, and BS tests. It is better in the GCD test with the JB goodness-of-fit test for k. At the 0.01 level of significance, the CS goodness-of-fit test applied with the GCD test cannot reject the null hypothesis for 4 scenarios.

Reaction of the tests for MT, MM, and LE generators is similar to that of the WH generator. According to the proportions of success, success rates of the MT generator are satisfactory for 16 and 32-bit numbers for all sequence lengths; and that of the MM generator is very satisfactory for 16 and 32-bit numbers for all sequence lengths, and 8-bit numbers for short and long sequence lengths. Success proportions of LE, KT, and KT02 generators are high for 16 and 32-bit numbers for all sequence lengths, and 8-bit numbers for short sequences. The BS test rejects randomness of the KT02 generator for 8-bit numbers for all sequence lengths at the 0.05 level of significance. However, it cannot reject the null hypothesis for 8-bit numbers for all sequence lengths for $\alpha = 0.01$.

For 64-bit numbers, only the random walk excursion test with AD and KS goodness-of-fit tests cannot reject the null hypothesis for all RNG's. None of the random walk tests decides randomness of RNG's for 128-bit numbers. RNG's pass TBT, Achi, and GCD for k with AD, KS, and CS goodness-of-fit tests for almost all sequence lengths. This situation decreases the proportion of success for 64 and 128-bit numbers. This result would be due to the conservativeness of random walk height, random walk expansion tests, and GCD test with the Jarque-Bera goodness-of-fit test for higher bit lengths.

Summary

Statistical analysis of randomness of a cryptographic random number generator is a critical and necessary task to make use of the generator in cryptographic applications. Many cryptographic randomness tests are available for this task including recently proposed ones. Although there are several alternatives, the chi-square test is frequently employed within these cryptographic randomness tests as a goodness-of-fit test. In this regard, this article describes the package **CryptRndTest** that conducts frequently used and newly proposed 8 cryptographic randomness tests along with Anderson-Darling, Kolmogorov-Smirnov, chi-square, and Jarque-Bera goodness-of-fit tests. Totally, package **CryptRndTest** runs 21 tests. It also provides auxiliary functions for the calculation of the greatest common divisor, sequence of partial quotients resulting from the greatest common divisor operation, the base conversion from 2 to 10 and vice versa, and the Stirling numbers of the second kind. All of these auxiliary functions also work with long integers by the use of multi-precision floating point numbers.

In addition to the description of package **CryptRndTest**, random number generators available in R are tested by 21 cryptographic randomness tests of **CryptRndTest** under various combinations of sequence and bit-lengths. Implementation performance of package **CryptRndTest** is also revealed by the numerical application.

The limitations of the package are mostly related to the memory and CPU capacities of the computer used to run functions of **CryptRndTest**. Because, increasing bit-length considerably decreases the implementation speed of tests working over integers, this can also be seen as a limitation for high bit-lengths.

Acknowledgments

This work is fully supported by The Scientific and Technological Research Council of Turkey (TUBITAK) under Grant No. 114F249 of ARDEB-3001 programme. Authors wish to thank three anonymous reviewers and the editor for constructive comments that improved the quality and clarity of the article.

Bibliography

P. Alcover, A. Guillamon, and M. Ruiz. A new randomness test for bit sequences. *Informatica*, 24: 339–356, 2013. [p234, 239, 240]

W. Bleick and P. Wang. Asymptotics of Stirling numbers of the second kind. *Proceedings of the American Mathematical Society*, 42:575–580, 1974. [p240]

R. Brown, D. Eddelbuettel, and D. Bauer. Dieharder: A random number test suite (version 3.31.1). URL: http://www.phy.duke.edu/ rgb/General/dieharder.php, 2014. [Online; accessed 25-February-2014]. [p233]

F. Caeiro and A. Mateus. randtests: Testing randomness in R. https://CRAN.R-project.org/package=randtests, 2014. Online; accessed 2016-02-25. [p234]

H. Demirhan. *CryptRndTest: Statistical Tests for Cryptographic Randomness*, 2016. URL https://CRAN.R-project.org/package=CryptRndTest. R package version 1.2.1. [p234]

H. Demirhan and N. Bitirim. Statistical testing of cryptographic randomness. *Journal of Statisticians: Statistics and Actuarial Sciences*, 9:1–11, 2016. [p233]

A. Doganaksoy, C. Calik, F. Sulak, and M. Turan. New randomness tests using random walk. In *Proceedings of National Cryptology Symposium II*, Turkey, 2006. [p234, 238, 239]

S. Doroshenko and B. Ryabko. The experimental distinguishing attack on RC4. Cryptology ePrint Archive, Report 2006/070, 2006. http://eprint.iacr.org/. [p237]

S. Doroshenko, A. Fionov, A. Lubkin, V. Monarev, and B. Ryabko. On ZK-crypt, book stack, and statistical tests. Cryptology ePrint Archive, Report 2006/196, 2006. http://eprint.iacr.org/. [p237]

D. Eddelbuettel and R. Brown. RDieHarder: An R interface to the dieharder suite of random number generator tests. http://CRAN.R-project.org/package=RDieHarder, 2014. [Online; accessed 16-June-2015]. [p233]

J. Hernandez, J. Sierra, and A. Seznec. The SAC test: A new randomness test, with some applications to PRNG analysis. *In:Proceedings of the International Conference Computational Science and Its Applications*, pages 960–967, 2004. [p234]

M. Hofert, I. Kojadinovic, M. Maechler, and J. Yan. *copula: Multivariate Dependence with Copulas*, 2015. URL http://CRAN.R-project.org/package=copula. R package version 0.999-14. [p240]

D. Knuth. *The Art of Computer Programming, Volume 2 / Seminumerical Algorithms*. Addison-Wesley, Reading, Massachusetts, 1 edition, 1969. [p233]

D. Knuth. *The Art of Computer Programming, Volume 2 / Seminumerical Algorithms*. Addison-Wesley, Reading, Massachusetts, 2 edition, 1981. [p233]

D. Knuth. *The Art of Computer Programming, Volume 2 / Seminumerical Algorithms*. Addison-Wesley, Reading, Massachusetts, 3 edition, 1998. [p233]

P. L'Ecuyer and P. Hellekalek. Random number generators: Selection criteria and testing. *Random and Quasi-Random Point Sets Lecture Notes in Statistics*, 138:223–265, 1998. [p233]

P. L'Ecuyer and R. Simard. TestU01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, 33:Article 22, 2007. [p233]

P. L'Ecuyer and R. Simard. TestU01: A C library for empirical testing of random number generators – user's guide, compact version. http://www.iro.umontreal.ca/~simardr/testu01/guideshorttestu01.pdf, 2014. [Online; accessed 24-February-2014]. [p233]

A. Lucas, I. Scholz, R. Boehme, S. Jasson, and M. Maechler. *gmp: Multiple Precision Arithmetic*, 2014. URL https://CRAN.R-project.org/package=gmp. R package version 0.5-12. [p240]

M. Maechler. Arbitrarily accurate computation with R: Package Rmpfr. https://CRAN.R-project.org/web/packages/Rmpfr/vignettes/Rmpfr-pkg.pdf, 2011. [Online; accessed 18-December-2015]. [p243]

M. Maechler. *Rmpfr: R MPFR – Multiple Precision Floating-Point Reliable*, 2015. URL https://CRAN.R-project.org/package=Rmpfr. R package version 0.6-0. [p235, 243]

G. Marsaglia. Diehard: A battery of tests of randomness. http://stat.fsu.edu/~geo/diehard.html, 1996. [Online; accessed 25-February-2014]. [p233]

G. Marsaglia and W. Tsang. Some difficult to pass tests of randomness. *Journal of Statistical Software*, 7 (3):1–9, 2002. [p233, 235, 236, 237, 238, 241, 242]

M. Mascagni and A. Srinivasan. Algorithm 806: SPRNG: A scalable library for pseudorandom number generation. *ACM Transactions on Mathematical Software*, 26:436–461, 2000. [p233]

U. Maurer. A universal statistical test for random bit generators. *Journal of Cryptology*, 5:89–105, 1992. [p234]

B. McCullough. A review of TestU01. *Journal of Applied Econometrics*, 21:677–682, 2006. [p233]

M. Ruetti. *A Random Number Generator Test Suite for the C++ Standard-Diploma Thesis*. Institute for Theoretical Physics, ETH Zurich, 2004. [p233]

A. Rukhin. Testing randomness: A suite of statistical procedures. *Theory of Probability and Its Applications*, 45:111–132, 2001. [p233]

A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo. A statistical test suite for random and pseudorandom number generators for cryptographic applications. http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf, 2010. [Online; accessed 17-June-2015]. [p233, 234]

B. Ryabko and V. Monarev. Using information theory approach to randomness testing. *Journal of Statistical Planning and Inference*, 133:95–110, 2005. [p234, 236, 237, 241]

B. Ryabko, V. Stognienko, and Y. Shokin. A new test for randomness and its application to some cryptographic problems. *Journal of Statistical Planning and Inference*, 123:365–376, 2004. [p234, 235]

J. Sadique, U. Zaman, and R. Ghosh. Review on fifteen statistical tests proposed by NIST. *Journal of Theoretical Physics and Cryptography*, 1:18–31, November 2012. [p233]

F. Scholz and A. Zhu. *kSamples: K-Sample Rank Tests and Their Combinations*, 2016. URL https://CRAN.R-project.org/package=kSamples. R package version 1.2-3. [p236]

A. Signorell. DescTools: Tools for descriptive statistics. https://CRAN.R-project.org/package=DescTools, 2015. Online; accessed 2016-02-25. [p234]

J. Soto. Statistical testing of random number generators. In *Proceedings of the 22nd National Information Systems Security Conference*, USA, 1999. National Institute of Standards and Technology. [p233]

M. Sýs and Z. Říha. Faster randomness testing with the NIST statistical test suite. In R. Chakraborty et al., editors, *Security, Privacy, and Applied Cryptography Engineering Lecture Notes in Computer Science*, pages 272–284, Portugal, 2014. Springer. [p233]

M. Sýs, P. Švenda, M. Ukrop, and V. Matyáš. Constructing empirical tests of randomness. In A. H. Mohammad S. Obaidat and P. Samarati, editors, *SECRYPT 2014 Proceedings of the 11th International Conference on Security and Cryptography*, pages 229–237, Portugal, 2014. SCITEPRESS – Science and Technology Publications. [p233]

N. Temme. Asymptotic estimates of Stirling numbers. *Studies in Applied Mathematics*, 89:233–243, 1993. [p240]

A. Trapletti and K. Hornik. *tseries: Time Series Analysis and Computational Finance*, 2015. URL http://CRAN.R-project.org/package=tseries. R package version 0.10-34. [p238]

I. Vattulainen, T. Ala-Nissila, and K. Kankaala. Physical models as tests of randomness. *Physical Review Engineering*, 52:3205–3213, 1995. [p233]

J. Walker. ENT – A pseudorandom number sequence test program. https://www.fourmilab.ch/random/, 2014. [Online; accessed 25-February-2014]. [p233]

Haydar Demirhan
Hacettepe University
Department of Statistics 06800 Beytepe Ankara
Turkey
and
RMIT University
School of Science
Mathematical Sciences
3001 Melbourne Victoria
Australia
haydarde@hacettepe.edu.tr
haydar.demirhan@rmit.edu.au

Nihan Bitirim
Council of Higher Education
06800 Cankaya Ankara
Turkey

scmamp: Statistical Comparison of Multiple Algorithms in Multiple Problems

by Borja Calvo and Guzmán Santafé

Abstract Comparing the results obtained by two or more algorithms in a set of problems is a central task in areas such as machine learning or optimization. Drawing conclusions from these comparisons may require the use of statistical tools such as hypothesis testing. There are some interesting papers that cover this topic. In this manuscript we present **scmamp**, an R package aimed at being a tool that simplifies the whole process of analyzing the results obtained when comparing algorithms, from loading the data to the production of plots and tables.

Comparing the performance of different algorithms is an essential step in many research and practical computational works. When new algorithms are proposed, they have to be compared with the state of the art. Similarly, when an algorithm is used for a particular problem, its performance with different sets of parameters has to be compared, in order to tune them for the best results.

When the differences are very clear (e.g., when an algorithm is the best in all the problems used in the comparison), the direct comparison of the results may be enough. However, this is an unusual situation and, thus, in most situations a direct comparison may be misleading and not enough to draw sound conclusions; in those cases, the statistical assessment of the results is advisable.

The statistical comparison of algorithms in the context of machine learning has been covered in several papers. In particular, the tools implemented in this package are those presented in Demšar (2006); García and Herrera (2008); García et al. (2010). Another good review that covers, among other aspects, the statistical assessment of the results in the context of supervised classification can be found in Santafé et al. (2015).

Existing tools

Some of the methods presented in the referred papers are well known procedures that are included in classical statistics tools. As an example, *p-value* correction methods such as Holm (Holm, 1979) or omnibus tests such as Friedman's (Friedman, 1937) are implemented in R's base package. However, other methods are neither so well known nor trivial to implement. Worth highlighting is Bergmann and Hommel's procedure (Bergmann and Hommel, 1988) to correct the *p-values* when all the algorithms are compared pair-wise (see García and Herrera, 2008, page 2681).

There are tools that implement some of the methods included in this package. The first one is KEEL (Alcalá-Fdez et al., 2008), a Java toolbox that includes a module for the statistical assessment of the results obtained in a given experiment. However, although it can be used independently from the rest of the toolbox, its GUI offers only a limited combination of methods and any other analysis requires programming within Java.

As an alternative to the KEEL GUI, STATService 2.0 (Parejo et al., 2012) provides a web service to perform statistical analysis of multiple algorithms using KEEL's code. Along the same lines we have STAC (Rodríguez-Fdez et al., 2015), a Python web service that allows running different types of parametric and non-parametric tests using a simple interface and its own implementation of the methods.

The goal of **scmamp** is to provide a simple pipeline that allows any researcher to load their complete set of results, analyze them and produce the material needed for publication (tables and plots).

Under some circumstances we may be interested in analyzing the results in different groups of problems. An example of such a situation are the results presented in Blum et al. (2015), where the behavior of a set of algorithms was tested in problems of different size and complexity[1]. In order to deal with these kinds of problems, conversely to other existing tools, the package offers the possibility of subsetting the results, which can be handy when the problems can be subdivided into different groups (e.g., based on their size).

Another advantage of the **scmamp** package over other existing implementations is that the functions that perform the analyses accept additional user-defined test and correction functions, increasing

[1] As we will show later, part of these results are available in **scmamp** as an example of the type of results matrix used by the package.

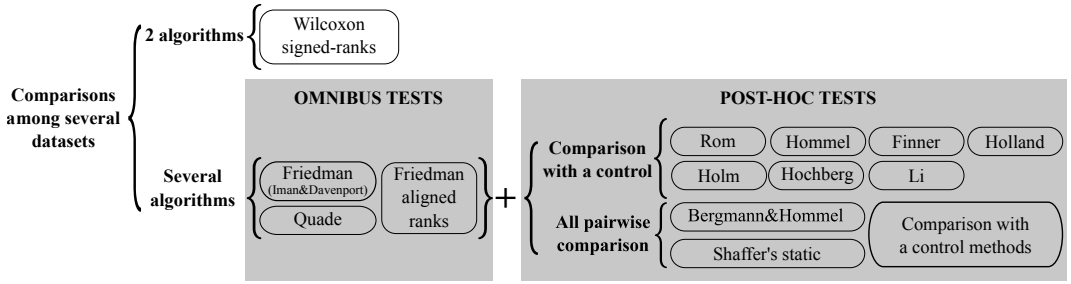


Figure 1: Recommended statistical test for different scenarios.

the flexibility of the analysis. Moreover, all the correction methods included in the **stats** package through the `p.adjust` function can be directly used in the **scmamp** package.

Finally, we mention that although KEEL and STATService generate tables to be directly used in publications, they do not generate plots. In our package we have included two functions to graphically represent the results of the comparison. Moreover, performing the analysis in R allows the user to easily create his/her own plots.

Brief overview of the statistical tests and general recommendations

Several publications (Demšar, 2006; García and Herrera, 2008; García et al., 2010; Santafé et al., 2015) have stated a basic classification of general machine learning scenarios and the associated statistical tests which are appropriate for each situation. This package is mainly focused in the comparison of multiple algorithms in multiple datasets. However, due to the flexibility of the implemented functions, it can also be adapted to other situations. Figure 1 summarizes the most common situations when evaluating machine learning algorithms.

Comparisons of two algorithms among a set of different problems are common in the literature in order to decide between two competitors. The estimated scores for each algorithm on each problem are independent. However, as they may be obtained from different application domains (or problems with different characteristics), it is highly debatable whether they can be averaged over in order to obtain a meaningful overall estimation of the algorithm's performance. Consequently, non-parametric methods such as Wilcoxon signed-rank are usually recommended (Demšar, 2006).

On the other hand, in order to compare multiple algorithms in multiple problems, the general recommended methodology is as follows. First, we apply an omnibus test to detect if at least one of the algorithms performs differently than the others. Second, if we find a significant difference, then we apply a pair-wise test with the corresponding *post-hoc* correction for multiple comparisons. The Friedman test with Iman and Davemport extension is probably the most popular omnibus test, and it is usually a good choice when comparing more than five different algorithms. By contrast, when comparing five or fewer different algorithms, Friedman aligned ranks and the Quade test are more powerful alternatives (García et al., 2010).

Regarding *post-hoc* tests, the choice depends on the pair-wise comparisons: comparison with a control or all pair-wise comparisons. When comparing all the algorithms with a control, Bonferroni is the most simple but the least powerful procedure and, thus, it is not recomended in practice. By contrast, Hommel and Rom are the two most powerful procedures but they are also more complex than other methods. Alternatively, Finner is a simple procedure and it is the next with the highest power (see García et al., 2010). Nevertheless, except for Bonferroni, in practice there are not very big differences in the power of the *post-hoc* tests. Therefore, in general, Finner's method is a good choice due to its simplicity and power.

Similarly, when an all pair-wise comparison is conducted, those procedures to perform comparisons with a control can be used. In this case, the Nemenyi's test is the most simple but less powerful alternative and it is not usually recommended in practice. Alternatively, specific methods for all pair-wise comparison have a higher power. The one with highest power is Bergmann and Hommel, but it is a complex and computationally expensive method. The **scmamp** package optimizes this method for up to nine algorithms by having some pre-computed operations. Therefore, comparing more than nine algorithms with the Bergmann and Hommel procedure is unfeasible in practice. In those situations, Shaffer's static method or even other more simple procedures such as Finer and Holm are recommended.

Brief examples

In this section we will illustrate the use of the package in three different situations. For a more detailed discussion on the use of the different functions the reader is referred to the package's vignettes. These may be accessed with the command `browseVignettes('scmamp')`.

The first example of use comes from García and Herrera (2008). Actually, we will use the set of results presented in that paper, which are included in the package in the variable `data.gh.2008`. These results collect the performance of a number of supervised classification algorithms in a set of 30 datasets. The goal of the study is comparing the different algorithms and determining which outperforms which. For more details, the reader is referred to García and Herrera (2008).

```
> library('scmamp')
> head(data.gh.2008)

            C4.5 k-NN(k=1) NaiveBayes Kernel   CN2
Abalone*   0.219     0.202      0.249  0.165 0.261
Adult*     0.803     0.750      0.813  0.692 0.798
Australian 0.859     0.814      0.845  0.542 0.816
Autos      0.809     0.774      0.673  0.275 0.785
Balance    0.768     0.790      0.727  0.872 0.706
Breast     0.759     0.654      0.734  0.703 0.714
```

The goal is analyzing all the pair-wise comparisons. Therefore, the first hypothesis to test is whether all the algorithms perform equally or, in contrast, some of them have a significantly different behavior. Then, all the differences are tested for every pair of algorithms and the resulting *p-values* are corrected. There are different ways to report the results, depending on the *post-hoc* analysis. A very intuitive tool is Demsar's critical difference plots. Although easy to interpret, these plots are based on the Nemenyi test, which is a very conservative one. There are other alternatives that can be used with more powerful methods (such as Bergmann and Hommel's correction). In this example we will use the `drawAlgorithmGraph` function which creates a graph based on the *p-values* corrected by any method.

First, we check the differences using the Iman and Davenport omnibus test.

```
> imanDavenportTest(data.gh.2008)

Iman Davenport's correction of Friedman's rank sum test

data:  data.gh.2008
Corrected Friedman's chi-squared = 14.3087, df1 = 4, df2 = 116,
p-value = 1.593e-09
```

The *p-value* shown above denotes that there is at least one algorithm that performs differently than the rest and, therefore, we can proceed with the *post-hoc* analysis of the results. There are several alternatives to perform this analysis, but we will focus on two. The first alternative is the Nemenyi test. Although, this test is not a recommended choice in practice since it is very conservative and has a low power, it is shown in this example because its associated plot is quite illustrative.

```
> nm <- nemenyiTest(data.gh.2008)
> nm

Nemenyi test

data:  data.gh.2008
Critical difference = 1.1277, k = 5, df = 145

> nm$diff.matrix

          C4.5  k-NN(k=1) NaiveBayes    Kernel        CN2
[1,]  0.000000 -1.1500000 -0.1000000 -2.233333 -1.0166667
[2,] -1.150000  0.0000000  1.0500000 -1.083333  0.1333333
[3,] -0.100000  1.0500000  0.0000000 -2.133333 -0.9166667
[4,] -2.233333 -1.0833333 -2.1333333  0.000000  1.2166667
[5,] -1.016667  0.1333333 -0.9166667  1.216667  0.0000000
```

This procedure determines the *critical difference*. Any two algorithms whose performance difference is greater that the critical difference are regarded as significantly different. As can be seen in the code

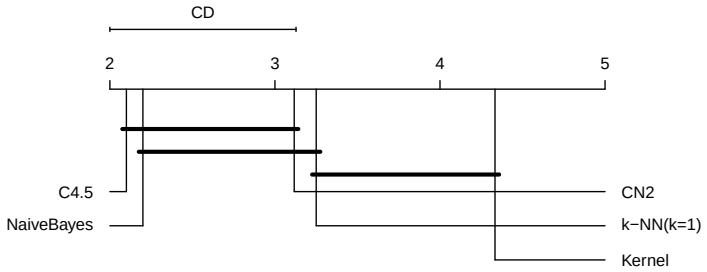


Figure 2: Example of critical difference plot.

above, the differences between every pair of algorithms is stored in the `diff.matrix` element of the result. The Nemenyi test has the advantage of having an associated plot to represent the results of the comparison. This plot can be obtained as follows.

```
> plotCD(results.matrix = data.gh.2008, alpha = 0.05)
```

The result can be seen in Figure 2. In this plot each algorithm is placed on an axis according to its average ranking. Then, those algorithms that show no significant differences are grouped together using a horizontal line. The plot also shows the size of the critical difference required for considering two algorithm as significantly different.

The second alternative shown in this example is the Friedman *post-hoc* test with Bergmann and Hommel's correction. Both steps can be carried out in a single line of code.[2]

```
> test.res <- postHocTest(data = data.gh.2008, test = 'friedman', correct = 'bergmann')
> test.res

$summary
       C4.5 k-NN(k=1) NaiveBayes    Kernel       CN2
[1,] 0.7797    0.6791     0.7565 0.5693667 0.7285333

$raw.pval
                   C4.5   k-NN(k=1)   NaiveBayes       Kernel         CN2
C4.5                 NA 0.004848763 8.064959e-01 4.486991e-08 0.012763008
k-NN(k=1)  4.848763e-03          NA 1.011233e-02 7.963489e-03 0.743971478
NaiveBayes 8.064959e-01 0.010112334           NA 1.736118e-07 0.024744672
Kernel     4.486991e-08 0.007963489 1.736118e-07           NA 0.002880485
CN2        1.276301e-02 0.743971478 2.474467e-02 2.880485e-03          NA

$corrected.pval
                   C4.5  k-NN(k=1)   NaiveBayes       Kernel        CN2
C4.5                 NA 0.02909258 1.000000e+00 4.486991e-07 0.03828902
k-NN(k=1)  2.909258e-02         NA 3.185396e-02 3.185396e-02 1.00000000
NaiveBayes 1.000000e+00 0.03185396           NA 1.041671e-06 0.03828902
Kernel     4.486991e-07 0.03185396 1.041671e-06           NA 0.01152194
CN2        3.828902e-02 1.00000000 3.828902e-02 1.152194e-02         NA
```

The above code runs the *post-hoc* test, computing the raw *p-value* for each pair of algorithms. These *p-values* are also corrected for multiple testing using Bergman and Hommel's correction. Additionally, a summary with the average values of each algorithm over all the dataset is obtained.

The package offers two ways of presenting the results obtained in the analysis. On the one hand, we can create a LaTeX table using the function `writeTabular`. This function includes parameters to control several aspects of the table. For more details on its use the reader is referred to the vignette covering the data loading and manipulation. The table generated can be seen in Table 1.

```
> # LaTeX formated: Significances highlighted in bold
> bold <- test.res$corrected.pval < 0.05
> bold[is.na(bold)] <- FALSE
> writeTabular(table = test.res$corrected.pval, format = 'f', bold = bold,
```

[2]These steps can be carried out independently using the functions implemented in the package. For more information the reader is referred to the package documentation.

	C4.5	k-NN(k=1)	NaiveBayes	Kernel	CN2
C4.5	n/a	**0.029**	1.000	**0.000**	**0.038**
k-NN(k=1)	**0.029**	n/a	**0.032**	**0.032**	1.000
NaiveBayes	1.000	**0.032**	n/a	**0.000**	**0.038**
Kernel	**0.000**	**0.032**	**0.000**	n/a	**0.012**
CN2	**0.038**	1.000	**0.038**	**0.012**	n/a

Table 1: Example of table generated by the package.

```
+              hrule = 0, vrule = 0)

\begin{tabular}{|l|llll|}
\hline
 & C4.5 & k-NN(k=1) & NaiveBayes & Kernel & CN2 \\
\hline
C4.5 & n/a & {\bf 0.029} & 1.000 & {\bf 0.000} & {\bf 0.038} \\
k-NN(k=1) & {\bf 0.029} & n/a & {\bf 0.032} & {\bf 0.032} & 1.000 \\
NaiveBayes & 1.000 & {\bf 0.032} & n/a & {\bf 0.000} & {\bf 0.038} \\
Kernel & {\bf 0.000} & {\bf 0.032} & {\bf 0.000} & n/a & {\bf 0.012} \\
CN2 & {\bf 0.038} & 1.000 & {\bf 0.038} & {\bf 0.012} & n/a \\
\hline
\end{tabular}
```

On the other hand, the results can be shown in a graph that represents the algorithms that show no significant differences as connected nodes.

```
> average.ranking <- colMeans(rankMatrix(data.gh.2008))
> drawAlgorithmGraph(pvalue.matrix = test.res$corrected.pval,
+                    mean.value = average.ranking)
```

In the graph, shown in Figure 3, we can see that, according to the test, there are no significant differences within the pairs C4.5/NaiveBayes and kNN/CN2. Compared with the critical difference plot, this method is able to detect more differences (for example, the Nemenyi test does not detect significant differences between the kNN and kernel algorithms).

For the second example we will use a dataset where the problems can be subdivided into groups. The dataset shows the results obtained by eight different algorithms used to find large independent sets in graphs. The performance of the algorithms is evaluated in a number of randomly generated graphs of different size and density. Thus the results matrix we will use contains in each row the result obtained by each algorithms for a given problem. This dataset is part of the results in Blum et al. (2015), and it is also included in the package.

```
> head(data.blum.2015)

  Size Radius FruitFly Shukla Ikeda Turau Rand1 Rand2 FrogCOL FrogMIS
1 1000  0.049      223    213   214   214   214   212     246     226
2 1000  0.049      224    207   209   216   205   211     241     219
3 1000  0.049      219    206   215   214   209   213     243     221
4 1000  0.049      227    208   218   218   215   219     251     230
5 1000  0.049      231    218   210   212   211   217     243     239
6 1000  0.049      230    214   214   208   211   206     246     229
```

The first two columns indicate the size and the density of the random graph while the last eight columns contain the results obtained by each algorithm. Although we could directly use the data loaded in the package, we use this example to briefly show how data can be loaded from one or more files.

Depending on how the experimentation is conducted, we may end up with a number of files, each containing part of the full result. Moreover, it can happen that the information we need is not only in the file content, but also in its name. The package includes a set of functions whose goal is simplifying this task of combining results. Here we will show how the data can be loaded from a set of example files distributed with the package. For further information about how data can be loaded the reader is referred to the corresponding package vignette.

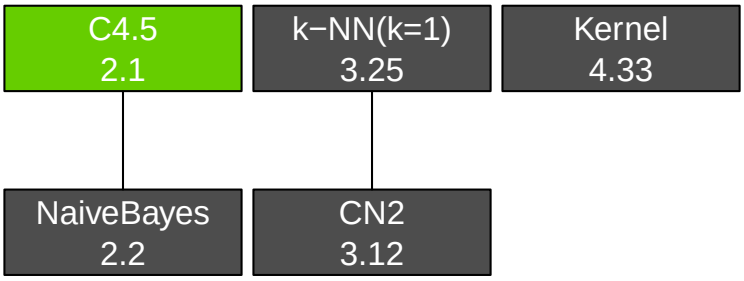


Figure 3: Example of algorithm graph.

```
> dir <- paste(system.file('loading_tests', package = 'scmamp'),
+              'experiment_files', sep ='/')

[1] "rgg_size_1000_r_0.049_FrogCOL.out"
[2] "rgg_size_1000_r_0.049_FrogMIS.out"
[3] "rgg_size_1000_r_0.049_FruitFly.out"
[4] "rgg_size_1000_r_0.049_Ikeda.out"
[5] "rgg_size_1000_r_0.049_Rand1.out"

> pattern <- 'rgg_size_([0-9]*)_r_(0.[0-9]*)_([a-z,A-Z,1,2]*).out'
> var.names <- c('Size', 'Radius', 'Algorithm')
> dataset <- readExperimentDir(directory = dir, names = var.names,
+                              fname.pattern = pattern, alg.var.name = 'Algorithm',
+                              value.col = 'Evaluation', col.names = 'Evaluation')
> head(dataset)

  Size Radius FrogCOL FrogMIS FruitFly Ikeda Rand1 Rand2 Shukla Turau
1 1000  0.049     246     226      223   214   214   212    213   214
2 1000  0.049     241     219      224   209   205   211    207   216
3 1000  0.049     243     221      219   215   209   213    206   214
4 1000  0.049     251     230      227   218   215   219    208   218
5 1000  0.049     243     239      231   210   211   217    218   212
6 1000  0.049     246     229      230   214   211   206    214   208
```

In order to extract information from the file names, the name structure has to be defined as a regular expression. In the above code we can see that there are three variables that are extracted from the file names: `Size`, `Radius` and `Algorithm`. Note that the latter does not appear in the final dataset, as it has been used to generate the different columns in the table.

In this dataset there are 30 problems for each combination of size and radius. For each problem the table contains the results obtained with eight algorithms. In this case, we want to compare all the algorithms with the reference one, FrogCOL. Thus, we will compare, using an Iman and Davenport test, the average performance of the algorithms for each combination of size and radius. First, we compute the mean values using the `summarizeData` function. Then, we run the test.

```
> dataset.means <- summarizeData(dataset, group.by = c('Radius', 'Size'),
+                                fun = mean, na.rm = TRUE)
> imanDavenportTest(data.gh.2008)

Iman Davenport's correction of Friedman's rank sum test

data:  data.gh.2008
Corrected Friedman's chi-squared = 14.3087, df1 = 4, df2 = 116,
p-value = 1.593e-09
```

The small *p-value* obtained in the test indicates that there is strong statistical evidence to state that at least one algorithm performs differently than the rest. Thus, a *post-hoc* test (Friedman + Finner's correction) can be conducted to detect differences by pairs.

```
> res <- postHocTest(data = dataset.means, algorithms = 3:10, test = 'friedman'
+                    correct = 'finner' , control = 'FrogCOL')
```

FrogCOL	FrogMIS	FruitFly	Ikeda	Rand1	Rand2	Shukla	Turau
132.6	125.5	105.9*	116.1*	116.3*	116.6*	117.1*	116.2*

Table 2: Rendering of a LaTeX table generated with the `writeTabular` function.

The table with the results can be generated as follows (the result can be seen in Table 2):

```
> best.res <- res$summary == max(res$summary)
> stat.diff <- res$corrected.pval < 0.05
> stat.diff[is.na(stat.diff)] <- FALSE
> writeTabular(table = res$summary, format = 'f', bold = best.res, mark = stat.diff,
+              digits = 1)
```

Finally, for the third example we will use the same dataset from Blum et al. (2015). In this example we want to compare the algorithms separately for every value of *Radius* given a fixed *Size*. The functions in **scmamp** can be also used for these kinds of comparisons. In this case we will use the Wilcoxon test with the *p-values* corrected using Finner's method and, then, a LaTeX table will be generated. In this table the best results will be highlighted in bold font and those without significant differences will be identified with a superscript.

First, filter the data.

```
> sub.dataset <- filterData(data = dataset, condition = 'Size==1000'
+                           remove.cols = 'Size')
```

Now, run the comparison.

```
> res <- postHocTest(data = sub.dataset, group.by = 'Radius', test = 'wilcoxon'
+                    correct = 'finner', control = 'FrogCOL')
> res$corrected.pval[1:5,1:6]

  Radius FrogCOL     FrogMIS     FruitFly       Ikeda       Rand1
1  0.049      NA 6.07021e-05 0.0000607021 6.07021e-05 6.07021e-05
2  0.058      NA 6.07021e-05 0.0000607021 6.07021e-05 6.07021e-05
3  0.067      NA 6.07021e-05 0.0000607021 6.07021e-05 6.07021e-05
4  0.076      NA 6.07021e-05 0.0071920496 6.07021e-05 6.07021e-05
5  0.085      NA 6.07021e-05 0.0005028761 6.07021e-05 6.07021e-05
```

Finally, generate the table. The boolean matrices needed for highlighting the results can be created manually or using the `booleanMatrix` function included in the package (the result is rendered in Table 3).

```
> tab <- res$summary
> best.res <- booleanMatrix(data = tab[, -1], find = 'max', by = 'row')
> best.res <- cbind(FALSE, best.res)
> no.diff <- booleanMatrix(data = res$corrected.pval[, -1], find = 'gt' , th = 0.05)
> no.diff <- cbind(FALSE, no.diff)
> no.diff[is.na(no.diff)] <- FALSE
> digits <- c(3, rep(1, 8))
> writeTabular(table = tab, format = 'f', bold = best.res, mark = no.diff,
+              hrule = 0, vrule = 1, print.row.names = FALSE, digits = digits)
```

Conclusions

The **scmamp** package has been designed with the goal of simplifying the statistical analysis of the results obtained in comparisons of algorithms in multiple problems. With a few lines of code, the users can load and analyse the data and format the result for publication. This document is a brief introduction to the package. For further details on the use of the functions the reader is referred to the documentation and, particularly, to the vignettes of the package.

Radius	FrogCOL	FrogMIS	FruitFly	Ikeda	Rand1	Rand2	Shukla	Turau
0.049	**247.7**	227.6	226.3	213.2	212.7	214.5	212.4	211.9
0.058	**189.9**	176.9	174.9	162.0	161.5	163.5	162.9	162.7
0.067	**151.9**	140.6	142.2	130.4	129.8	129.6	130.8	129.9
0.076	**122.9**	114.1	117.8	104.6	105.3	104.9	105.3	105.3
0.085	**102.4**	94.3	99.4	85.9	86.7	87.1	87.4	86.7
0.094	85.5	79.3	**85.8**	72.9	72.6	72.3	74.2	72.9
0.103	74.2	68.1	**75.6***	62.8	63.2	62.3	63.2	62.1
0.112	64.4	58.2	**66.9**	54.1	54.2	54.4	54.5	54.8
0.121	56.5	51.3	**59.5**	47.4	47.0	47.6	48.0	47.2
0.134	**47.5**	43.1	24.3	40.1	39.9	40.1	40.8	40.1

Table 3: Another rendering of a LaTeX table generated with the `writeTabular` function.

Bibliography

J. Alcalá-Fdez, L. Sánchez, S. García, M. J. del Jesus, S. Ventura, J. M. Garrell, J. Otero, C. Romero, J. Bacardit, V. M. Rivas, J. C. Fernández, and F. Herrera. KEEL: A software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, 13(3):307–318, 2008. [p248]

B. Bergmann and G. Hommel. Improvements of general multiple test procedures for redundant systems of hypotheses. In *Multiple Hypotheses Testing*, volume 70, pages 100–115. Springer Berlin Heidelberg, 1988. [p248]

C. Blum, B. Calvo, and M. J. Blesa. FrogCOL and FrogMIS: New decentralized algorithms for finding large independent sets in graphs. *Swarm Intelligence*, 9:205–227, 2015. [p248, 252, 254]

J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006. [p248, 249]

M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937. [p248]

S. García, A. Fernández, J. Luengo, and F. Herrera. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, 180(10):2044–2064, 2010. [p248, 249]

S. García and F. Herrera. An extension on 'Statistical comparisons of classifiers over multiple data sets' for all pairwise comparisons. *Journal of Machine Learning Research*, 9:2677–2694, 2008. [p248, 249, 250]

S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6: 65–70, 1979. [p248]

J. A. Parejo, J. García, A. Ruiz-Cortés, and J. C. Riquelme. STATService: Herramienta de análisis estadístico como soporte para la investigación con metaheurísticas. In *Actas del VIII Congreso Expañol sobre Metaheurísticas, Algoritmos Evolutivos y Bio-inspirados*, 2012. [p248]

I. Rodríguez-Fdez, A. Canosa, M. Mucientes, and A. Bugarín. STAC: A web platform for the comparison of algorithms using statistical tests. In *Proceedings of the 2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2015. [p248]

G. Santafé, I. Inza, and J. A. Lozano. Dealing with the evaluation of supervised classification algorithms. *Artificial Intelligence Review*, 44(4):467–508, 2015. [p248, 249]

Borja Calvo
Department of Computer Science and Artificial Intelligence
University of the Basque Country (UPV/EHU)
Manuel de Lardizabal, 1
20018, Donostia (Spain)
borja.calvo@ehu.eus

Guzmán Santafé
Department of Statistics and Operations Research
Public University of Navarre

Campus de Arrosadía
31006, Pamplona (Spain)
guzman.santafe@unavarra.es

keyplayer: An R Package for Locating Key Players in Social Networks

by Weihua An and Yu-Hsin Liu

Abstract Interest in social network analysis has exploded in the past few years, partly thanks to the advancements in statistical methods and computing for network analysis. A wide range of the methods for network analysis is already covered by existent R packages. However, no comprehensive packages are available to calculate group centrality scores and to identify key players (i.e., those players who constitute the most central group) in a network. These functionalities are important because, for example, many social and health interventions rely on key players to facilitate the intervention. Identifying key players is challenging because players who are individually the most central are not necessarily the most central as a group due to redundancy in their connections. In this paper we develop methods and tools for computing group centrality scores and for identifying key players in social networks. We illustrate the methods using both simulated and empirical examples. The package **keyplayer** providing the presented methods is available from Comprehensive R Archive Network (CRAN).

Introduction

Interest in social network analysis has grown rapidly in the past few years. This was due partly to the advancements in statistical methods and computing for network analysis and partly to the increasing availability of social network data (e.g., network data generated by social media). A wide range of the methods for network analysis is already covered by R packages such as **network** (Butts, 2008b), **sna** (Butts, 2008a), **igraph** (Csardi and Nepusz, 2006), **statnet** (Handcock et al., 2008), **RSiena** (Ripley et al., 2013), etc. However, none of these packages provides a comprehensive toolbox to calculate group centrality measures and to identify key players, who constitute the most central group, in a network. Determining the key players in a network is very important because many social and health interventions rely on key players to facilitate the intervention. For example, Kelly et al. (1991) and Latkin (1998) trained peer leaders as educators to promote HIV prevention. Campbell et al. (2008) and An (2015) used peer leaders to facilitate smoking prevention. Borgatti (2006) and Ressler (2006) suggested removing key figures among terrorists to most widely disrupt terrorism. More examples of this sort can be found in Valente and Pumpuang (2007), Banerjee et al. (2013), etc. Identifying key players is challenging because players who are individually the most central are not necessarily the most central as a group due to redundancy in their connections. In a seminal paper, Borgatti (2006) pointed out the problem and proposed methods for identifying key players in social networks.

To the best of our knowledge, the `keyplayer` function in **UCINET** (Borgatti et al., 2002) is the first implementation of the methods detailed in Borgatti (2006). It has evolved from a separate add-on to **UCINET** to a built-in function **UCINET**. In this paper, we present the **keyplayer** package (An and Liu, 2016) in R, which differs from the `keyplayer` function in **UCINET** in several aspects. (1) Unlike the `keyplayer` function in **UCINET** which is only applicable to binary networks, **keyplayer** in R can be used for both binary and weighted networks. (2) The **keyplayer** package includes more centrality measures for choosing key players than what is currently available in the `keyplayer` function in **UCINET**. (3) **keyplayer** provides better integration with other open-source packages in R. Overall, the `keyplayer` function in **UCINET** is useful for researchers who are more familiar with **UCINET** and would like to utilize other functionalities provided by **UCINET**, whereas **keyplayer** is designed for users who are more familiar with R and who plan to do more computational work.

The **influenceR** package (Simon and Aditya, 2015) aims to provide calculations of several node centrality measures that were previously unavailable in other packages, such as the constraint index (Burt, 1992) and the bridging score (Valente and Fujimoto, 2010). It can also be used to identify key players in a network. But in comparison to **keyplayer**, it utilizes only one centrality metric when selecting key players whereas **keyplayer** includes eight different metrics. Also, **influenceR** currently works only for undirected networks whereas **keyplayer** works for both undirected and directed networks. Both packages provide parallel computation. **influenceR** relies on OpenMP for parallel computation whereas **keyplayer** utilizes the base package **parallel** which is readily available in R. Last, **influenceR** focuses on computing centrality measures at the node level whereas **keyplayer** is more interested in providing centrality measures at the group level. Overall, **keyplayer** provides more comprehensive functionalities for calculating group centrality measures and for selecting key players.

The algorithm for identifying key players in package **keyplayer** essentially consists of three steps. First, users choose a metric to measure centrality in a network. Second, the algorithm (specifically the

$$W = \begin{bmatrix} 0 & 1 & 3 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 1 & 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 3 \\ 0 & 2 & 0 & 0 & 0 \end{bmatrix}$$

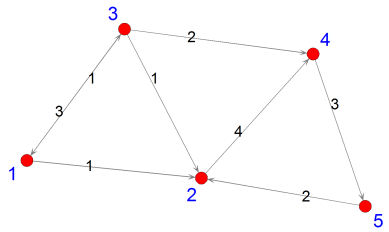


Figure 1: An adjacency matrix (left) and the corresponding network graph (right).

kpcent function) will randomly pick a group of players and measure their group centrality. Third, the algorithm (specifically the kpset function) will select the group of players with the highest group centrality as the desired key players. In general, users only need to employ the kpset function by specifying a centrality metric and the number of key players to be selected. The function will return a set of players who are the most central as a group. We also make the auxiliary function kpcent available. If users specify a centrality metric and the indices of a group of players, this function will return the centrality score of the specified group. Thus the two functions can be used for two purposes: selecting key players or measuring group centrality.

The paper proceeds as follows. First, we review centrality measures at the individual level. Then we present methods for measuring centrality at the group level. After that, we present a greedy search algorithm for selecting key players and outline the basic structure and the usage of the main function kpset in package **keyplayer**. To illustrate the methods and the usage of the package, we use a simulated network as well as an empirical example based on the friendship network among managers in a company. Last, we summarize and point out directions for improving the package in the future.

Measuring individual centrality

We first review the definitions of centrality measures at the individual level. For conciseness, we provide the definitions based on weighted networks, where the weight of a tie takes a continuous value and usually measures the strength of the connection between two nodes. The definitions naturally incorporate binary networks where the weight of a tie can only be one or zero, indicating the presence or absence of a connection (Freeman, 1978; Wasserman and Faust, 1994; Butts, 2008a).

Figure 1 shows an example of a simulated network. On the left is the adjacency matrix of the network. On the right is the network graph. Thinking of it as a friendship network, we can see that the strength of friendship between node 1 and node 3 is conceived differently by node 1 and node 3. The former assigns it a weight of 3 while the latter assigns it a weight of 1. We will use this example to illustrate the centrality measures. Calculations of four centrality measures (i.e., degree, closeness, betweenness, and eigenvector centralities) at the individual level are done using the **sna** package (Butts, 2008a). Calculations of four other individual level centralities and all group level centralities are done using our package **keyplayer**. We would like to clarify at this point that our package does not depend on **sna**. We use **sna** here just for the sake of the example.

Degree centrality

Degree centrality is defined as follows (Freeman, 1978; Butts, 2008a):[1]

$$D_i = \sum_j w_{ij} + \sum_j w_{ji}, \tag{1}$$

where w_{ij} represents the tie status from node i to node j. Thus the first term indicates the outgoing connections from node i (i.e., outdegree) and the second term the incoming connections to node i (i.e., indegree). Degree centrality measures a node's direct connectedness with other nodes in a network. After loading the adjacency matrix in R, we can type the following in R to get the degree, indegree, and outdegree measures for the simulated network.

[1] It may be worth noting that Freeman (1978) distinguishes absolute and relative measures of centrality. The definition here is based on Butts (2008a) and only considers the absolute number of connections.

```
> W <- matrix(c(0, 1, 3, 0, 0, 0, 0, 0, 4, 0, 1, 1, 0, 2, 0, 0, 0, 0, 0, 3,
+               0, 2, 0, 0, 0), nrow = 5, ncol = 5, byrow = TRUE)
> library(sna)
> degree(W, ignore.eval = FALSE) # For binary networks, set ignore.eval = TRUE.
[1] 5 8 7 9 5
> degree(W, ignore.eval = FALSE, cmode = "indegree")
[1] 1 4 3 6 3
> degree(W, ignore.eval = FALSE, cmode = "outdegree")
[1] 4 4 4 3 2
```

Closeness centrality

One version of the closeness centrality is due to Gil and Schmidt (1996):

$$C_i = \frac{\sum_j d_{ij}^{-1}}{n-1}, \tag{2}$$

where d_{ij} is the shortest path (i.e., geodistance) between nodes i and j. Closeness centrality usually reflects a node's capability of quickly reaching other nodes. In the above example, the tie status indicates friendship strength. The larger the value, the stronger is the friendship. To have the shortest path correspond to the strongest friendship, we need to transform the tie status, for example, by taking the inverse, before calculating the closeness centrality.

```
> A <- W
> A[W != 0] <- 1 / W[W != 0] # Inverse the non-zero tie status
> closeness(A, ignore.eval = FALSE, cmode = "suminvdir")
[1] 1.5142857 1.4285714 1.3000000 1.0500000 0.8333333
## For undirected networks, set cmode = "suminvundir".
```

Betweenness centrality

Betweenness centrality is defined as follows (Butts, 2008a):

$$B_i = \sum_{jk} \frac{g_{jk}^i}{g_{jk}}, \tag{3}$$

where g_{jk} is the number of shortest paths between nodes j and k, and g_{jk}^i is the number of those paths that pass node i. In the case of $g_{jk} = 0$, the corresponding contribution to the betweenness score is zero. Betweenness centrality usually measures a node's brokerage power in a network. We can get the betweenness centrality for the simulated network as follows.

```
> betweenness(A, ignore.eval = FALSE, cmode = "directed")
[1] 0 1 2 3 1
```

Eigenvector centrality

Eigenvector centrality defines a node's centrality as a weighted average of the centrality of its neighbors (Bonacich, 1972; Butts, 2008a):

$$E_i = \frac{1}{\lambda} \sum_j w_{\mathrm{ij}} E_j. \tag{4}$$

In matrix notations, this is equivalent to

$$\lambda E = WE,$$

where W represents the adjacency matrix and λ the largest eigenvalue of the above equation. Eigenvector centrality measures the extent to which a node is connected to important alters. To get the eigenvector centrality for the adjacency matrix A, we type the following in R:

```
> evcent(A, gmode = "digraph", ignore.eval = FALSE, use.eigen = TRUE)
[1] 0.5000000+0i 0.0000000+0i 0.8660254+0i 0.0000000+0i 0.0000000+0i
```

where `gmode = "digraph"` indicates that the input is a directed network and `use.eigen = TRUE` requests using the robust `eigen` function to calculate the eigenvectors. In this example, the eigenvector

centrality includes complex numbers, which are hard to interpret. Thus, to facilitate interpretation of the results, it is often a good idea to symmetrize the network first because symmetric matrices always have real eigenvalues. In the following, the symmetrization process first converts W to a binary matrix and then treats all ties as mutual ties.

```
> B <- symmetrize (W)
> evcent(B)
[1] 0.3505418 0.5590326 0.4699593 0.4699593 0.3505418
```

M-reach degree centrality

M-reach degree centrality generalizes the degree centrality by delimiting specific neighborhoods. Suppose the set of nodes that node i can reach via M steps is F and the set of nodes that can reach node i via M steps is H. Building on Borgatti (2006), we define the M-reach centrality as follows:

$$M_i = \sum_{j \in F} m_{ij} + \sum_{j \in H} m_{ji}, \tag{5}$$

where m_{ij} is 1 if $j \in F$ and m_{ji} is 1 if $j \in H$. The first term indicates the number of nodes that node i can reach in M steps. The second term indicates the number of nodes that can reach node i in M steps. By default, the matrix is binarized before calculating the centrality. Thus, in binary networks, the 1-reach degree centrality is the same as the degree centrality.

```
## Calculations of four other individual level centralities and all group level
## centralities are done by our package.
> library(keyplayer)
## M-reach centrality.
> mreach.degree(W, M = 1)
     outdegree indegree total
[1,]         2        1     3
[2,]         1        3     4
[3,]         3        1     4
[4,]         1        2     3
[5,]         1        1     2
```

M-reach closeness centrality

One way to refine the M-reach degree centrality is to use (the inverse of) geodistance to measure the tie status between nodes, just like how closeness centrality refines degree centrality. We define the M-reach closeness centrality as below:

$$MC_i = \frac{\sum_{j \in F} d_{ij}^{-1}}{d(n-1)} + \frac{\sum_{j \in H} d_{ji}^{-1}}{d(n-1)}, \tag{6}$$

where d_{ij} is the geodistance between nodes i and j, F and H are the set of nodes reachable from or to node i via M steps, respectively. d is the maximal of d_{ij}^{-1} across all pairs of i and j. The denominator helps normalize each of the two terms to be between zero and one. When M is infinity, M-reach closeness centrality approximates the Gil-Schmidt power index (Gil and Schmidt, 1996) and the cohesion centrality (Borgatti, 2006).

```
## As before, we first take the inverse of the tie status, making it correspond to
## distance.
> mreach.closeness(A)
     outdegree  indegree     total
[1,] 0.3785714 0.0625000 0.4410714
[2,] 0.3571429 0.3250000 0.6821429
[3,] 0.3250000 0.1875000 0.5125000
[4,] 0.2625000 0.5333333 0.7958333
[5,] 0.2083333 0.4232143 0.6315476
```

Fragmentation centrality

Fragmentation centrality measures the extent to which a network is fragmented after a node is removed from the network (Borgatti, 2006):

$$F_i = 1 - \frac{\sum_{j,k \neq i} d_{jk}^{-1}}{d \cdot (n-1)(n-2)}, \tag{7}$$

where d_{jk} is the geodistance between nodes j and k in the residual network after node i is removed and d the maximal of d_{jk}^{-1} across j and k. The second term in the above equation measures the cohesion of the residual network. Thus fragmentation centrality is the opposite of the cohesion centrality.

```
> fragment(A)
      fragment
[1,] 0.6365079
[2,] 0.7446429
[3,] 0.6733500
[4,] 0.8333333
[5,] 0.7250000
```

Diffusion centrality

Banerjee et al. (2013) proposed the diffusion centrality defined by the row sum of the following matrix:

$$S = \sum_{t=1}^{T} P^t, \tag{8}$$

where P is a probability matrix where P_{ij} measures the probability that node i can reach to node j.[2] Each cell in the matrix S measures the aggregate propensity that i can reach to j in T iterations. Each row sum of the matrix S indicates the importance of a node in disseminating information to alters (namely, the expected number of times that all alters receive the information from that node). Banerjee et al. (2014) show that as T goes to infinity the diffusion centrality can approximate the eigenvector centrality or the Katz-Bonacich centrality (Katz 1953; Bonacich 1987). In practice, Banerjee et al. (2013) used the diffusion centrality to study the word-of-mouth information dissemination. Now suppose we create a new adjacency matrix by treating non-zero elements in the original network as ones and we also know what q is. Then we can calculate the diffusion centrality as below.

```
## Create a new adjacency matrix.
> g <- W
> g[W != 0] <- 1
> g
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    1    0    0
[2,]    0    0    0    1    0
[3,]    1    1    0    1    0
[4,]    0    0    0    0    1
[5,]    0    1    0    0    0

## Create a matrix with the passing probabilities.
> q <- matrix(c(0, .2, .6, 0, 0, .1, 0, 0, .4, 0, .1, .1, 0, .4, 0, 0, .5, 0, 0, .3,
+               0, .4, 0, 0, 0), nrow = 5, ncol = 5, byrow = TRUE)
> q
     [,1] [,2] [,3] [,4] [,5]
[1,]  0.0  0.2  0.6  0.0  0.0
[2,]  0.1  0.0  0.0  0.4  0.0
[3,]  0.1  0.1  0.0  0.4  0.0
[4,]  0.0  0.5  0.0  0.0  0.3
[5,]  0.0  0.4  0.0  0.0  0.0

## Get the probability matrix and calculate diffusion centrality.
```

[2] In its original parametrization (Banerjee et al., 2013), $P = q \times g$, where q is a measure of the passing probability and g the adjacency matrix. For simplification and consistency with other centrality measures, our package asks users to input the probability matrix P directly. With information on q and the adjacency matrix, the probability matrix P can easily be calculated by their product. Below we show an example of how to accomplish this.

```
> P <- q * g
> P
     [,1] [,2] [,3] [,4] [,5]
[1,]  0.0  0.2  0.6  0.0  0.0
[2,]  0.0  0.0  0.0  0.4  0.0
[3,]  0.1  0.1  0.0  0.4  0.0
[4,]  0.0  0.0  0.0  0.0  0.3
[5,]  0.0  0.4  0.0  0.0  0.0

> diffusion(P, T = 5)
     diffusion
[1,]   1.50832
[2,]   0.59296
[3,]   0.99968
[4,]   0.48816
[5,]   0.63488
```

Measuring group centrality

Everett and Borgatti (1999) provide one of the first studies that explored ways to measure group centralities (mainly degree, closeness, and betweenness centralities) in undirected networks. In this paper, we provide more group centrality measures (including the eight ones outlined above) and extend the methods to both undirected and directed networks. The basic idea is to treat a group of nodes as a large pseudo-node. The key problem, then, is how to measure the tie status between the group and other outside nodes. For that purpose, we provide several criteria.

1. *Minimum.* According to the minimum criterion, the tie status between a group G and an outside node j is measured as the minimum of the (nonzero) edges between nodes in the group and the outside node.
$$E_{Gj} = \min_{g \in G} E_{gj}.$$
This criterion ensures that there is a shortest path between the group and the outside node. It is useful for calculating geodistance related measures. Hence, by default we use this criterion to calculate the group level measures of geodistance, closeness centrality, betweenness centrality, M-reach centralities, and fragmentation centrality.
2. *Maximum.* According to the maximum criterion, the tie status between a group G and an outside node j is measured as the maximum of the (nonzero) edges between nodes in the group and the outside node.
$$E_{Gj} = \max_{g \in G} E_{gj}.$$
This criterion is useful for measuring the maximal strength (not just the presence) of the connections between the group and the outside node. By default, we use the maximum criterion to compute the group level degree centrality and eigenvector centrality.
3. *Addition.* According to the addition criterion, the tie status between a group G and an outside node j is measured as the sum of the edges between nodes in the group and the outside node.
$$E_{Gj} = \sum_{g \in G} E_{gj}.$$
This criterion is useful for measuring the overall strength of the connections between the group and the outside node.
4. *Union.* The union criterion is designed for probability matrices. The tie status between a group G and an outside node j is measured as the probability that there is at least one path connecting the group with the outside node.
$$E_{Gj} = 1 - \prod_{g \in G} (1 - E_{gj}).$$
By default, we use the union criterion to calculate the group level diffusion centrality.

In the simulated network, suppose nodes 2 and 3 are grouped together. The connection between this group and node 4 according to the maximum criterion is $E_{G4} = 4$. Suppose we use matrix P as a probability network. Then the union criterion gives $E_{G4} = 1 - (1 - 0.4) \times (1 - 0.4) = 0.64$. The `contract` function automates these calculations and returns a reduced network matrix in which the node index will be re-ordered with the group as the last node.

```
## Group nodes 2 and 3 and measure the connections between the group and outside nodes
## using the maximum criterion.
> contract(W, c(2, 3), method = "max")
    1 4 5 set
1   0 0 0   3
4   0 0 3   0
5   0 0 0   2
set 1 4 0   0

## Group nodes 2 and 3 in the probability matrix and measure the connections
## between the group and outside nodes using the union criterion.
> contract(P, c(2, 3), method = "union")
      1    4   5  set
1   0.0 0.00 0.0 0.68
4   0.0 0.00 0.3 0.00
5   0.0 0.00 0.0 0.40
set 0.1 0.64 0.0 0.00
```

Once the tie status between the group and outside nodes is measured, we can use the centrality measures outlined above to calculate group centrality based on the reduced network. The `kpcent` function implements the calculations. Note that users do not need to explicitly deploy the `contract` function because `kpcent` automatically uses it in the background.

```
> kpcent(W, c(2, 3), type = "degree", cmode = "total", method = "max")
[1] 10
> kpcent(W, c(2, 3), type = "degree", cmode = "total", method = "min")
[1] 6
> kpcent(W, c(2, 3), type = "degree", cmode = "total", method = "min", binary = TRUE)
[1] 4
> kpcent(W, c(2, 3), type = "mreach.degree", cmode = "total", M = 1, binary = TRUE)
[1] 4
> kpcent(W, c(2, 3), type = "mreach.closeness", cmode = "total", M = 1, binary = TRUE)
[1] 1.333333
```

Selecting key players

Recall that the ultimate goal is to select the most central group of nodes from a network. This goal quickly becomes challenging as the network size grows. For example, to choose five key players out of 100 nodes, there are $\binom{100}{5} = 75,287,520$ possible combinations. To search for the optimal set of key players, in **keyplayer** we employ a greedy search algorithm as originally proposed in Borgatti (2006). We revised the algorithm in multiple ways to enhance its usability and efficiency. The basic idea of the algorithm is to select a set of nodes as seeds and then swap the selected nodes with unselected ones if the swap increases the group centrality. More specifically, the algorithm proceeds as follows.

Step 1. Select an initial candidate set C. The residual set is denoted as R.

Step 2. Update the candidate set C.

1) Start with the first node in C. Try to swap it with nodes in R sequentially (loop 1). Make the swap if it improves the centrality score of the resulting C. The number of iterations in loop 1 is defined as the number of iterations (over the nodes in the residual set).
2) Repeat loop 1 for each node in C sequentially (loop 2). The number of iterations in loop 2 is defined as the number of rounds (over the nodes in the candidate set).
3) Stop if (1) the change in C's centrality score is smaller than a specified threshold or (2) the process reaches a specified number of rounds (i.e., the number of iterations in loop 2).

Step 3. Return the final set C and the centrality score.

The function `kpset` implements the search algorithm. Its basic structure is shown below.

```
kpset(adj.matrix, size, type = "degree", M = Inf, T = ncol(adj.matrix),
      method = "min", binary = FALSE, cmode = "total", large = TRUE,
      geodist.precomp = NULL, seed = "top", parallel = FALSE, cluster = 2,
      round = 10, iteration = ncol(adj.matrix))
```

where the arguments are defined as follows.

- `adj.matrix`: Matrix indicating the adjacency matrix of the network or in the case of diffusion centrality a probability matrix.
- `size`: Integer indicating the target size of players.
- `type`: String indicating the type of centrality measure to be used. Should be one of `"degree"` for degree centrality, `"closeness"` for closeness centrality, `"betweenness"` for betweenness centrality, `"evcent"` for eigenvector centrality, `"mreach.degree"` for M-reach degree centrality, `"mreach.closeness"` for M-reach closeness centrality, `"fragment"` for fragment centrality, and `"diffusion"` for diffusion centrality.
- `M`: Positive number indicating the maximum geodistance between two nodes, above which the two nodes are considered disconnected. The default is `Inf`. The option is applicable to M-reach degree, M-reach closeness, and fragmentation centralities.
- `T`: Integer indicating the maximum number of iterations in the communication process. For diffusion centrality only. By default, `T` is the network size.
- `method`: Indication of which grouping criterion should be used. `"min"` indicates the "minimum" criterion and is suggested for betweenness, closeness, fragmentation, and M-reach centralities. `"max"` indicates the "maximum" criterion and is suggested for degree and eigenvector centralities. `"add"` indicates the "addition" criterion and is suggested for degree and eigenvector centralities as an alternative of `"max"`. `"union"` indicates the "union" criterion and is suggested for diffusion centrality. The default is `"min"`.
- `binary`: If `TRUE`, the input matrix is binarized. If `FALSE`, the edge values are considered. The default is `FALSE`.
- `cmode`: String indicating the type of centrality being evaluated. The option is applicable to degree and M-reach centralities. `"outdegree"`, `"indegree"`, and `"total"` refer to indegree, outdegree, and total degree, respectively. `"all"` reports all the above measures. The default is to report the total degree.
- `large`: Logical scalar. If `TRUE` (the default), the method implemented in **igraph** is used for computing geodistance and related centrality measures; otherwise the method in **sna** is used.
- `geodist.precomp`: Geodistance precomputed for the network to be analyzed (optional).
- `seed`: String indicating the seeding method or a vector of the seeds specified by user. If `"top"`, players with the highest individual centrality are used as the seeds. If `"random"`, seeds are randomly sampled. The default is `"top"` for efficiency.
- `parallel`: Logical scalar. IF `TRUE`, the parallel computation is activated. The default is `FALSE`.
- `cluster`: Integer indicating the number of CPU cores to be used for parallel computation.
- `round`: Integer indicating the "length" of search, namely, the number of loops over the nodes in the candidate set.
- `iteration`: Integer indicating the "width" of search in each round, namely, the number of loops over the nodes in the residual set.

The greedy algorithm converges fast, but sometimes can be trapped in a local optimum. To avoid this problem, it is recommended to run `kpset` several times with different seeds. To facilitate the search in large networks, users can employ parallel computation by specifying `parallel = TRUE` in `kpset`. During parallel computation, for each cluster and each iteration the algorithm randomly picks a node from the candidate set and the residual set, respectively, and swaps the two if it improves the centrality score of the candidate set. It repeats this process until exhausting the specified iterations and rounds and then combines the results from the clusters. The following code shows how to find two players who are the most central as a group in the simulated network.

```
## In terms of indegree.
> kpset(W, size = 2, type = "degree", cmode = "indegree", method = "max")
$keyplayers
[1] 3 4

$centrality
[1] 7

## In terms of indegree in the binarized network.
> kpset(W, size = 2, type = "degree", cmode = "indegree", binary = TRUE,
+       method = "max")
$keyplayers
[1] 2 4
```

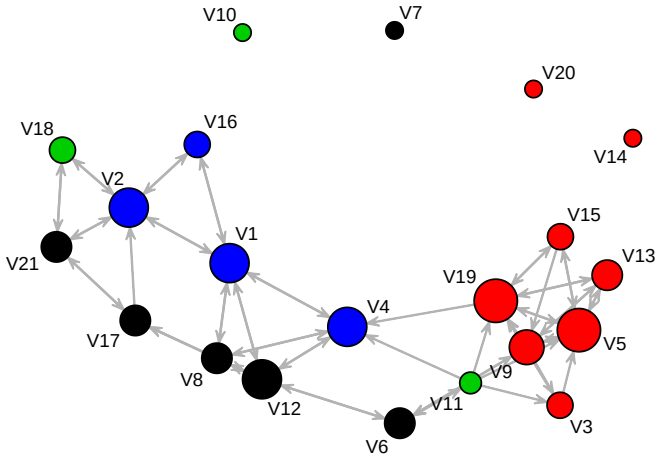


Figure 2: The friendship network of 21 managers in a high-tech company.

```
$centrality
[1] 3

## In terms of mreach.degree.
> kpset(W, size = 2, type = "mreach.degree", cmode = "indegree", M = 1,
+       binary = TRUE)
$keyplayers
[1] 2 4

$centrality
[1] 3

## In terms of mreach.closeness.
> kpset(A, size = 2, type = "mreach.closeness", cmode = "indegree", M = 1)
$keyplayers
[1] 3 4

$centrality
[1] 0.6944444

## In terms of indegree via parallel computation using 2 CPU cores.
> kpset(W, size = 2, type = "degree", cmode = "indegree", parallel = TRUE,
+       cluster = 2)
$keyplayers
[1] 3 4

$centrality
[1] 7
```

An empirical example

Below we use the friendship network of 21 managers in a high-tech company (Krackhardt, 1987) to illustrate the methods. The network graph is shown in Figure 2. Each node represents one manager. Each tie indicates a friendship nomination from one manager to the other. The nodes are colored according to the four departments the managers belong to. The size of each node is proportional to its degree. As it can be seen, friendships occur predominately within departments.

We first examine the individual centrality of the managers. To make a probability matrix for calculating the diffusion centrality, we multiply the original adjacency matrix by 0.1. The results are presented in Table 1. To facilitate reading the results, we marked the top centrality scores in red. Apparently, the most central manager identified varies with the centrality measure used. In terms of indegree, managers 5 and 19 each receive six friend nominations and are the most central. However, in terms of outdegree, managers 1, 9, 11, and 12 are the most central. In terms of closeness centrality,

ID	Indegree	Outdegree	Closeness	Between	Evcent	2-reach indegree	2-reach closeness	Fragment.	Diffusion (T=2)
1	5	**5**	0.44	60.83	0.32	11	0.40	0.71	**0.68**
2	5	4	0.39	29.58	0.17	8	0.33	0.70	0.52
3	2	2	0.34	0.00	0.14	6	0.20	0.68	0.28
4	5	3	0.38	46.25	0.25	**14**	**0.48**	0.70	0.44
5	**6**	4	0.40	13.00	0.28	7	0.33	0.69	0.55
6	3	3	0.43	79.75	0.26	10	0.33	0.74	0.45
7	0	0	0.00	0.00	0.00	0	0.00	0.65	0.00
8	3	4	0.41	5.67	0.29	9	0.30	0.69	0.56
9	4	**5**	0.45	47.08	0.31	8	0.30	0.70	0.66
10	0	0	0.00	0.00	0.00	0	0.00	0.65	0.00
11	1	**5**	**0.48**	20.50	0.32	3	0.10	0.69	0.66
12	5	**5**	0.47	**88.42**	**0.34**	11	0.40	**0.74**	0.68
13	3	3	0.37	1.17	0.23	7	0.25	0.68	0.43
14	0	0	0.00	0.00	0.00	0	0.00	0.65	0.00
15	2	3	0.37	1.17	0.23	6	0.20	0.68	0.43
16	2	2	0.33	0.00	0.13	8	0.25	0.68	0.29
17	3	3	0.39	27.83	0.17	8	0.28	0.69	0.42
18	2	2	0.30	0.00	0.07	5	0.18	0.68	0.27
19	**6**	4	0.42	44.17	0.26	7	0.33	0.70	0.53
20	0	0	0.00	0.00	0.00	0	0.00	0.65	0.00
21	3	3	0.35	8.58	0.11	7	0.25	0.68	0.39

Table 1: Centrality scores for the managers.

	KP1	KP2	KP3	Score
Indegree	2	5	12	14.00
Outdegree	2	5	12	12.00
Closeness	1	9	21	0.72
Betweenness	2	12	19	130.17
Evcent	2	11	12	0.58
2-reach indegree	1	7	19	15.00
2-reach closeness (indegree)	2	5	12	0.78
Fragmentation	1	6	17	0.82
Diffusion (T=2)	1	9	11	1.93

Table 2: The three managers who are the most central as a group.

manager 11 is the most central. In terms of betweenness centrality and eigenvector centrality, manager 12 is the most central, etc. Which centrality measure is suitable for selecting the most central player depends on the objectives. If the objective is to find a manager whose opinion is respected by most peers, then indegree can be a suitable measure. But if the objective is to spread the information most widely, then outdegree or closeness may be a better option.

Now suppose we want to find the three managers in this company who are the most central as a group. Table 2 lists the results according to different centrality measures. If indegree is the preferred centrality measure, then managers 2, 12, and 19 form the most central group. Together these three managers can connect to 14 other managers. Note that the three managers with the highest individual centrality do not constitute the most central group. The group indegree of managers 5, 19, and 1 (or 2 or 4) is no more than 10. Table 2 also shows that the most central group varies by centrality measure employed. Researchers are required to thoroughly thinkabout which centrality measure they should use in their specific context to select key players. In addition, sometimes there may be multiple sets of players which are equally central as a group. In such cases, which set is to be used may not make big difference in practice. But if examining these different sets is of interest, it is recommended to run `kpset` multiple times.

Summary

In this paper, we developed a comprehensive set of methods and tools for locating key players in social networks. In the future, the algorithms used may be improved by choosing seeds and swaps more strategically and by utilizing alternative optimization schemes such as simulated annealing.

Acknowledgments

The two authors contributed equally. Weihua An designed the study. Both authors contributed to writing the manuscript and developing the package. The authors thank Professor Bettina Grün and two anonymous reviewers for their helpful comments.

Bibliography

W. An. Multilevel meta network analysis with application to studying network dynamics of network interventions. *Social Networks*, 43:48–56, 2015. [p257]

W. An and Y.-H. Liu. *keyplayer: Locating Key Players in Social Networks*, 2016. URL https://CRAN.R-project.org/package=keyplayer. R package version 1.0.3. [p257]

A. Banerjee, A. G. Chandrasekhar, E. Duflo, and M. O. Jackson. Diffusion of microfinance. *Science*, 341 (6144):1–7, 2013. doi: 10.1126/science.1236498. [p257, 261]

A. Banerjee, A. G. Chandrasekhar, E. Duflo, and M. O. Jackson. Gossip: Identifying central individuals in a social network. Working Paper, 2014. [p261]

P. Bonacich. Factoring and weighting approaches to clique identification. *Journal of Mathematical Sociology*, 2(1):113–120, 1972. [p259]

P. Bonacich. Power and centrality: A family of measures. *American Journal of Sociology*, 5:1170–1182, 1987. [p261]

S. P. Borgatti. Identifying sets of key players in a network. *Computational, Mathematical and Organizational Theory*, 12:21–34, 2006. [p257, 260, 261, 263]

S. P. Borgatti, M. G. Everett, and L. C. Freeman. *UCINET for Windows: Software for Social Network Analysis*. Analytic Technologies, Harvard, MA, 2002. [p257]

R. S. Burt. *Structural Holes: The Social Structure of Competition*. Harvard University Press, Cambridge, MA, 1992. [p257]

C. T. Butts. Social network analysis with sna. *Journal of Statistical Software*, 24(6):1–51, 2008a. doi: 10.18637/jss.v024.i06. [p257, 258, 259]

C. T. Butts. network: A package for managing relational data in R. *Journal of Statistical Software*, 24(2): 1–36, 2008b. doi: 10.18637/jss.v024.i02. [p257]

R. Campbell, F. Starkey, J. Holliday, S. Audrey, M. Bloor, N. Parry-Langdon, R. Hughes, and L. Moore. An informal school-based peer-led intervention for smoking prevention in adolescence (ASSIST): A cluster randomised trial. *Lancet*, 371:1595–1602, 2008. [p257]

G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL http://igraph.org. [p257]

M. G. Everett and S. P. Borgatti. The centrality of groups and classes. *Journal of Mathematical Sociology*, 23(3):181–201, 1999. [p262]

L. C. Freeman. Centrality in social networks: Conceptual clarification. *Social Networks*, 1:215–239, 1978. [p258]

J. Gil and S. Schmidt. The origin of the Mexican network of power. In *Proceedings of the International Social Network Conference*, pages 22–25, Charleston, SC, 1996. [p259, 260]

M. S. Handcock, D. R. Hunter, C. T. Butts, S. M. Goodreau, and M. Morris. statnet: Software tools for the representation, visualization, analysis and simulation of network data. *Journal of Statistical Software*, 24(1):1–11, 2008. doi: 10.18637/jss.v024.i01. [p257]

L. Katz. A new status index derived from sociometric index. *Psychometrika*, 18(1):39–43, 1953. [p261]

J. A. Kelly, S. L. Janet, E. D. Yolanda, L. Y. Stevenson, A. C. Hauth, T. L. Brasfiel, S. C. Kalichman, J. E. Smith, and M. E. Andrew. HIV risk behavior reduction following intervention with key opinion leaders of population: An experimental analysis. *American Journal of Public Health*, 81:168–171, 1991. [p257]

D. Krackhardt. Cognitive social structures. *Social Networks*, 9:109–134, 1987. [p265]

C. A. Latkin. Outreach in natural settings: The use of peer leaders for HIV prevention among injecting drug users' networks. *Public Health Reports*, 113:S151–S159, 1998. [p257]

S. Ressler. Social network analysis as an approach to combat terrorism: Past, present, and future research. *Homeland Security Affairs*, 2(2):1–9, 2006. [p257]

R. Ripley, K. Boitmanis, and T. A. B. Snijders. *RSiena: Siena – Simulation Investigation for Empirical Network Analysis*, 2013. URL https://CRAN.R-project.org/package=RSiena. R package version 1.1-232. [p257]

J. Simon and K. Aditya. *influenceR: Software Tools to Quantify Structural Importance of Nodes in a Network*, 2015. URL https://CRAN.R-project.org/package=influenceR. R package version 0.1.0. [p257]

T. W. Valente and K. Fujimoto. Bridging: Locating critical connectors in a network. *Social Networks*, 2 (3):212–220, 2010. [p257]

T. W. Valente and P. Pumpuang. Identifying opinion leaders to promote behavior change. *Health Education and Behavior*, 34:881–896, 2007. [p257]

S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, Cambridge, NY, 1994. [p258]

Weihua An
Departments of Statistics and Sociology, Indiana University
752 Ballantine Hall, 1020 East Kirkwood Avenue, Bloomington, IN 47405, USA.
weihuaan@indiana.edu

Yu-Hsin Liu
Kelley School of Business, Indiana University
Hodge Hall 329, 1309 E 10th St, Bloomington, IN 47405, USA
yuhsliu@indiana.edu

SchemaOnRead: A Package for Schema-on-Read in R

by Michael J. North

Abstract **SchemaOnRead** is a CRAN package that provides an extensible mechanism for importing a wide range of file types into R as well as support for the emerging schema-on-read paradigm in R. The schema-on-read tools within the package include a single function call that recursively reads folders with text, comma separated value, raster image, R data, HDF5, NetCDF, spreadsheet, Weka, Epi Info, Pajek network, R network, HTML, SPSS, Systat, and Stata files. It also recursively reads folders (e.g., schemaOnRead("folder")), returning a nested list of the contained elements. The provided tools can be used as-is or easily customized to implement tool chains in R. This paper's contribution is that it introduces and describes the **SchemaOnRead** package and compares it to related R packages.

Introduction

SchemaOnRead is a CRAN package that provides an extensible mechanism for importing a wide range of file types into R as well as support for the emerging schema-on-read paradigm in R. The tools within the package include a single function call (e.g., `schemaOnRead("filename")`) that reads text (TXT), comma separated value (CSV), raster image (BMP, PNG, GIF, TIFF, and JPG)[1], R data (RDS), HDF5, NetCDF, spreadsheet (XLS, XLSX, ODS, and DIF), Weka Attribute-Relation File Format (ARFF), Epi Info (EPIINFO), Pajek network (NET), R network (PAJ), HTML, SPSS (SAV), Systat (SYS), and Stata (DTA) files. It also recursively reads folders (e.g., schemaOnRead("folder")), returning a nested list of the contained elements. The provided tools can be used as-is or easily customized to implement tool chains in R. This paper's contribution is that it introduces and describes the **SchemaOnRead** package and compares it to related R packages. In the sections that follow, this paper presents usage examples, discusses user defined processors, reviews the related work, explains the origin of the package name, summarizes the package contents, and then provides concluding thoughts.

Examples

A simple way to use SchemaOnRead is to conveniently load a file without needing to handle the specifics of the file format. In this case the result is a variable containing the file contents. Individual files can also be easily accessed without needing to known the specifics of the file format as below. The file contents can be accessed using the `xmlFile` variable. All of the source code and example data can be found at https://github.com/drmichaelnorth/SchemaOnRead.

```
library(SchemaOnRead)
xmlFile <- schemaOnRead("../inst/extdata/data.xml")
```

[1]Image processing applications are becoming increasingly popular for purposes such as pattern recognition and machine vision. These applications often read large numbers of files during their training and testing phases. Image file import has been added to **SchemaOnRead** to support this use case.

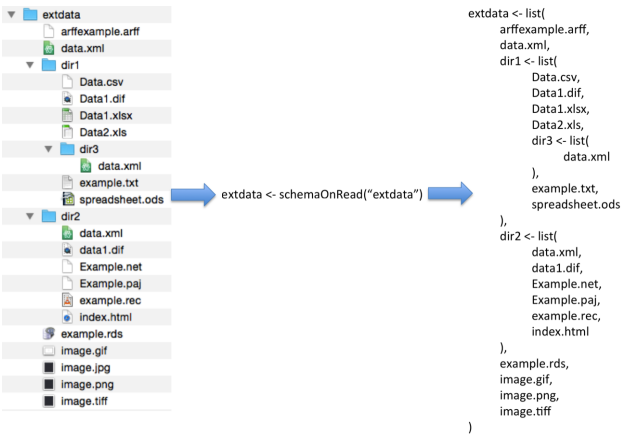


Figure 1: Reading a nested set of folders

Another way to use SchemaOnRead is to recursively load a folder. The result is a named list of elements for each entry in the folder's tree as shown in Figure 1. Sub-elements (e.g., files or subfolders) of a folder can be accessed using the R named list ($) operator followed by the sub-element name. An example showing how to read a folder tree starting in '../inst/extdata' is shown below.

```
library(SchemaOnRead)
results <- schemaOnRead("../inst/extdata")
```

In this case, the contents of the 'dir1/Data.csv' file within '../inst/extdata' is shown by accessing 'results$dir1$Data.csv' as needed. The path also provides the data provenance. Files or folders with names that do not conform to standard R variable naming requirements can be accessed using single quote notation (e.g., `results$'Nonconforming Name'`).

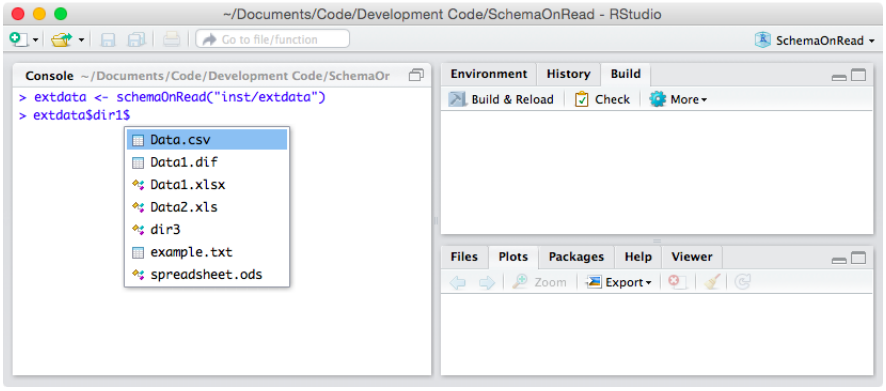


Figure 2: Using SchemaOnRead for convenient access to files and folders in RStudio

The resulting named list notation also provides convenient access to files and folders using integrated development environments for R that support automatic code completion. An RStudio (RStudio, 2015) example is shown in Figure 2.

The SchemaOnRead `verbose` flag can be used to trace a call's progress or diagnose issues as shown below.

```
library(SchemaOnRead)
folder <- schemaOnRead("../inst/extdata", verbose = TRUE)
```

Which produces the output:

```
schemaOnRead processing ../inst/extdata
schemaOnRead processing ../inst/extdata/arffexample.arff
schemaOnRead processing ../inst/extdata/data.xml
schemaOnRead processing ../inst/extdata/dir1
schemaOnRead processing ../inst/extdata/dir1/Data.csv
schemaOnRead processing ../inst/extdata/dir1/Data1.dif
schemaOnRead processing ../inst/extdata/dir1/Data1.xlsx
schemaOnRead processing ../inst/extdata/dir1/Data2.xls
schemaOnRead processing ../inst/extdata/dir1/dir3
schemaOnRead processing ../inst/extdata/dir1/dir3/data.xml
schemaOnRead processing ../inst/extdata/dir1/example.txt
schemaOnRead processing ../inst/extdata/dir1/spreadsheet.ods
schemaOnRead processing ../inst/extdata/dir2
schemaOnRead processing ../inst/extdata/dir2/data.xml
```

User Defined Processors

New processors can be defined to support user-specified processing. New processors are normally prepended to the front of the default list to allow them to take precedence while still allowing the standard processors to work if needed. Alternatively, a list of processors that just recursively scans folders can be found by calling the `schemaOnReadSimpleProcessors` function. User-specified processors can be added to this list to create a fully customized tool chain. An example showing how to create a simple files processor is given below.

```
## Load the needed library.
library(SchemaOnRead)

## Define a new processor.
newProcessor <- function(path, ...) {

  # Check the file existance and extensions.
  if (!SchemaOnRead::checkExtensions(path, c("xyz"))) return(NULL)

  ## As an example, attempt to read an XYZ file as a CSV file.
  read.csv(path, header = FALSE)

}

## Define a new processors list.
newProcessors <- c(newProcessor, SchemaOnRead::defaultProcessors())

# Use the new processors list.
schemaOnRead(path = "../inst/extdata", processors = newProcessors)
```

A more detailed example of a Microsoft Excel spreadsheet processor is shown below.

```
## Load the needed library.
library(SchemaOnRead)

## Define a new processor.
newSpreadsheetProcessor <- function(filePath = ".", ...) {

  # Check the file existance and extensions.
  if (!SchemaOnRead::checkExtensions(filePath, c("xls", "xlsx"))) return(NULL)

  # Read the workbook's worksheet names.
  worksheets <- readxl::excel_sheets(filePath)

  # Read the workbook's worksheets.
  workbook <- lapply(worksheets, readxl::read_excel, path = filePath)

  # Name the worksheets.
  names(workbook) <- worksheets

  # Return the results.
  workbook

}

## Define a new processors list.
newProcessors <- c(newSpreadsheetProcessor, SchemaOnRead::defaultProcessors())

# Use the new processors list.
schemaOnRead(path = "../inst/extdata", processors = newProcessors)
```

Related Work

Several R packages provide support for importing diverse file formats into R. Examples include **rio**, **readbitmap**, and **foreign**.

The **rio** package (Chan et al., 2015) is the closest in functionality to **SchemaOnRead**. **rio** provides file reading functions for a wide range of formats including text files, fixed format files, spreadsheet files (XLS, XLSX, ODS, and DIF),Stata, JSON, SPSS, Weka, Epi Info, serialized R objects, saved R objects, SAS, Minitab, Systat, shallow XML files, FORTRAN data files, and clipboard imports. **rio** supports a few file formats not imported by **SchemaOnRead** such as fixed format files, FORTRAN data files, and clipboard imports. **SchemaOnRead** similary offers several formats not supported by **rio** such as deep XML, BMP, JPEG, and PNG files. Unlike **SchemaOnRead**, **rio** includes functions for writing as well as reading. Unlike **rio**, **SchemaOnRead** includes functions for recursively reading directories and offers an interface that is easily extensible by end users.

The **foreign** package (R Core Team et al., 2015) provides functions for reading a range of file types including Weka, Epi Info, SPSS, Stata, Systat files. **SchemaOnRead** uses **foreign** for reading these types of files. Unlike **SchemaOnRead**, **foreign** uses different user function calls to select the format of the file being imported. Unlike **foreign**, **SchemaOnRead** provides recursive reading of folders, is designed to be easily extended by end users to new file formats, and checks file extensions to determine formats.

The **readbitmap** package (Jefferis, 2015) provides functions for reading BMP, JPEG, and PNG files. **SchemaOnRead** uses **readbitmap** for reading BMP, JPEG and PNG files. Unlike **SchemaOnRead**, **readbitmap** uses magic numbers rather than extensions to identify file formats [2]. Unlike **readbitmap**, **SchemaOnRead** provides recursive reading of folders and is designed to be easily extended by end users to new file formats.

Why "SchemaOnRead?"

Schema-on-read (Deutsch, 2013), (Mendelevitch, 2013), (Jacobsohn and Delurey, 2014) is an agile approach to data storage and retrieval that defers investments in data organization until production queries need to be run by working with data directly in native form. Schema-on-read functions have been implemented in a wide range of analytical systems including Hadoop (Hadoop Team, 2015), (Schau, 2015), Splunk (Bitincka et al., 2012), Apache Spark (Spark Team, 2015), Apache Flink (Markl, 2014), and even relational databases (Liu and Gawlick, 2015). It is also possible to use machine learning tools to extract schemas from source data (Yeh et al., 2013).

The R Package SchemaOnRead

The **SchemaOnRead** R package defines four public functions:

- `schemaOnRead(path = ".",processors = defaultProcessors(),verbose = FALSE)` processes the given path using the provided lsit of processors optinally printing its progress on the console.
- `defaultProcessors()` returns a complete list of built-in processors in the recommended execution order.
- `simpleProcessors()` returns a minimal list of built-in processors in the recommended execution order.
- `checkExtensions(path = ".",extensions = NULL)` returns true if the path exists and, if an extensions list is provided, the extension of the path is in extensions list.

The `schemaOnRead` function is used to read source material (e.g., files and folders).

The SchemaOnRead package uses a recursive implementation. The initial user function call, `schemaOnRead` iterates over the given list of processors, invoking each in turn until one returns a non-null value. Processors are sequentially invoked in the order given by the input list, scanning from index number one upwards. Processing continues as long as each processor returns null. The results from the first processor to return a non-null value is stored as the content for the entry and processing of that entry stops. All of the results are stored in a named list. The order of the resulting list is the order given by the file system. The variable names are taken from the entry names (e.g., file or folder names). Files or folders with names that do not conform to standard R variable naming requirements can be accessed using single quote notation (e.g., `results$'Nonconforming Name'`).

An example processor for Microsoft Excel spreadsheets is shown below. In this example, the entry identified by the `path` string is checked to see if it exists as a file. If it does, then the file name is extension is checked. If it matches then the processor attempts to read the file.

```
## Define the XLS and XLSX spreadsheet file processor.
schemaOnReadProcessXLSandXLSXFile <- function(path = ".",
        processors = schemaOnReadDefaultProcessors(), verbose = FALSE) {

        ## Check the given path.
        if ((file.exists(path)) &&
                ((tolower(tools::file_ext(path)) == "xls") ||
```

[2] Magic numbers (Wikipedia, 2015) are special values in files that represent the file format. Magic numbers are commonly stored as special values encoded in file headers and footers. The first two bytes of JPEG files in hexidecmial are FF and D8 and the last two bytes are FF and D9. The first six bytes of GIF files in hexdecimal are 47, 49, 46, 38, 37, and 61 (GIF87a in ASCII) or 47, 49, 46, 38, 39, and 61 (GIF89a in ASCII). The first eight bytes of PNG files in hexidecmial are 89, 50, 4E, 47, 0D, 0A, 1A, and 0A) which, in part, spells PNG in ASCII.

```
                 (tolower(tools::file_ext(path)) == "xlsx"))) {

                ## Create the results holder.
                results <- list()

                ## Attempt to read the file.
                workbook <- XLConnect::loadWorkbook(path)

                ## Scan the worksheets.
                for (worksheet in XLConnect::getSheets(workbook)) {

                        ## Define the variable name.
                        variable <- gsub("[^[:alnum:].]", "_", worksheet)
                        while (eval(parse(
                                text = paste("exists(\"results$", variable, "\")",
                                        sep = "")))) {
                                variable <- paste(variable, "_A", sep = "")
                        }

                        ## Setup the processing command.
                        command <- paste("results$", variable,
                                " <- XLConnect::readWorksheet",
                                "(workbook, sheet = worksheet)", sep = "")

                        ## Evaluate the processing command.
                        eval(parse(text = command))

                }

                ## Return the results.
                return(results)

        } else {

                ## Return the default value.
                return(NULL)

        }

}
```

The main goal of a processor is to read each acceptable entry into R in an easily usable format. Examples include the production of lists and data frames. The main output of SchemaOnRead is thus intended to be a nested tree of lists, with data frames in some of the leaves the tree. The first example does this by scanning the worksheets in a given workbook and converting each into a data frame. The result is a list of data frames with each data frame entry identified using the name of the corresponding worksheet. Note that the worksheet names are checked to insure that they correspond to valid R variable names for convenient user access.

The postconditions for each processor are that the processor or one of its descendants either successfully processes the entry and returns a non-null result or fail to process the entry and return null. If the entry is successfully processed then SchemaOnRead will perform no further processing on the item. If the item was not successfully processed then SchemaOnRead will use its remaining processors list to attempt to process the entry.

Several special processors are defined for SchemaOnRead. These include processors for nonexistent entries, directories, and entries of unknown types.

The `schemaOnReadProcessEntryDoesNotExist` processor returns null if the given entry exists and returns the value `"Entry Does Not Exist"` if not. It is meant to be the first processor in most lists to intercept nonexistent entries before they waste execution time in other processors. Occasionally, special processing may needed for nonexistent entries so these processors should run first.

The `schemaOnReadProcessDirectory` processor handles directories as previously discussed. It is intended to be the second processor to run in normal lists.

The `schemaOnReadProcessDefaultFile` processor accepts all entries that exist and returns the `"File Type Unknown"` string. It normally runs last to insure a value for unrecognized file types.

SchemaOnRead includes predefined two processing lists. The default processing list is used for SchemaOnRead entry processing. The simple processing list provides an easy starting point for user-defined processor lists.

Twenty-one unit tests are defined for the SchemaOnRead package. These tests are implemented using the **testthat** R package (Wickham, 2015). The current version of SchemaOnRead passes all of the defined tests.

Summary

As we have discussed, schema-on-read is a powerful new option for data storage and retrieval. Schema-on-read functions have been implemented in a wide range of analytical systems, most notably Hadoop. SchemaOnRead uses R's flexible data representations to provide transparent and convenient support for the schema-on-read paradigm in R. This paper's contribution is that it introduces and describes the **SchemaOnRead** package and compares it to related R packages.

Acknowledgements

Argonne National Laboratory's work was supported under U.S. Department of Energy contract DE-AC02-06CH11357.

Bibliography

L. Bitincka, A. Ganapathi, and S. Zhang. Experiences with workload management in splunk. In *Workshop on Management of Big Data Systems*, pages 25–30, 2012. [p272]

C. H. Chan, G. C. H. Chan, T. J. Leeper, and C. Gandrud. CRAN rio Package, Version 0.2. https://cran.r-project.org/web/packages/rio/index.html, 2015. [p271]

T. Deutsch. Why is schema on read so useful? http://www.ibmbigdatahub.com/blog/why-schema-read-so-useful, 2013. [p272]

Hadoop Team. Apache hadoop. http://hadoop.apache.org, 2015. [p272]

M. Jacobsohn and M. Delurey. How the data lake works. https://www.boozallen.com/content/dam/boozallen/documents/Data_Lake.pdf, 2014. [p272]

G. Jefferis. CRAN readbitmap Package, Version 0.1-4. https://cran.r-project.org/web/packages/readbitmap/index.html, 2015. [p272]

Z. H. Liu and D. Gawlick. Management of flexible schema data in rdbmss - opportunities and limitations for nosql. In 7^{th} *Biennial Conference on Innovative Data Systems Research*, 2015. [p272]

V. Markl. Breaking the chains: On declarative data analysis and data independence in the big data era. In *Proceedings of the VLDB Endowment*, volume 7, pages 1730–1733, 2014. [p272]

O. Mendelevitch. Apache hadoop and data agility. http://hortonworks.com/blog/hadoop-data-agility/, 2013. [p272]

R Core Team, R. Bivand, V. J. Carey, S. DebRoy, S. Eglen, R. Guha, N. Lewin-Koh, M. Myatt, B. Pfaff, B. Quistorff, F. Warmerdam, S. Weigand, and Free Software Foundation, Inc. CRAN foreign Package, Version 0.8-66. https://cran.r-project.org/web/packages/foreign/index.html, 2015. [p272]

RStudio. RStudio. https://www.rstudio.com, 2015. [p270]

A. Schau. Schema-on-read in action. http://blog.cask.co/2015/03/schema-on-read-in-action/, 2015. [p272]

Spark Team. Apache spark. http://spark.apache.org, 2015. [p272]

H. Wickham. CRAN testthat Package, Version 0.10.0. https://cran.r-project.org/web/packages/testthat/index.html, 2015. [p274]

Wikipedia. Magic number (programming). http://en.wikipedia.org/wiki/Magic_number_(programming), 2015. [p272]

E. Yeh, J. Niekrasz, and D. Freitag. Unsupervised discovery and extraction of semi-structured regions in text via self-information. In *Proceedings of the 2013 Workshop on Automated Knowledge Base Construction*, pages 103–107, 2013. [p272]

Michael J. North
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL 60439 USA
north@anl.gov

Crowdsourced Data Preprocessing with R and Amazon Mechanical Turk

by Thomas J. Leeper

Abstract This article introduces the use of the Amazon Mechanical Turk (MTurk) crowdsourcing platform as a resource for R users to leverage crowdsourced human intelligence for preprocessing "messy" data into a form easily analyzed within R. The article first describes MTurk and the **MTurkR** package, then outlines how to use **MTurkR** to gather and manage crowdsourced data with MTurk using some of the package's core functionality. Potential applications of **MTurkR** include construction of manually coded training sets, human transcription and translation, manual data scraping from scanned documents, content analysis, image classification, and the completion of online survey questionnaires, among others. As an example of massive data preprocessing, the article describes an image rating task involving 225 crowdsourced workers and more than 5500 images using just three **MTurkR** function calls.

Introduction

Often people use R because it is extensible, robust, and free. It can do many things, but doing those many things generally requires data structures that can be handled computationally. Yet sometimes R users are faced with messy data that are not "R-ready." Examples include: when working with handwritten survey responses, digitized texts that cannot be read by optical character recognition, images, etc. Other times an analyst may face machine-readable data that requires human interpretation to categorize, translate, or code the data, e.g., someone wishing to build an automated classifier needs a human-categorized training set to test their implementation.

In such cases, making the leap from these raw data to R data structures can entail considerable human labor. Such needs for human labor in data preprocessing has provoked interest in online crowdsourcing platforms (Schmidt, 2010; Chen et al., 2011) to bring human intelligence to tasks that cannot be easily accomplished through computation alone. This paper describes the use of **MTurkR** (Leeper, 2016) to leverage the Amazon Mechanical Turk (MTurk) crowdsourcing platform to bring human intelligence into R. The article begins by laying out the need for occasional human intelligence in data preprocessing, then describes MTurk and its vocabulary, and introduces **MTurkR**.

The need for human intelligence

Some data cannot be computationally preprocessed. Other data can be handled computationally only with difficulty. In these cases, data preprocessing can be a time consuming and expensive task because of the human intelligence required. Archetypal needs for this kind of human intelligence include the collection of data which cannot be automated (e.g., unstructured or malformed web data), transcription of files into machine-readable data (e.g., audio, images, or handwritten documents scanned as PDFs), tasks that are laborious to translate from an R-readable but non-computable data structure into a format that can be readily analyzed (e.g., text answers to free-response survey questions), or massive-scale machine readable data that require human interpretation (e.g., the data used in generating a training set for supervised learning algorithms).

Due to the manual nature of these tasks, preprocessing such data can become challenging, especially as the size of the dataset increases. Crowdsourcing these data preprocessing needs is therefore one way to obtain the scalable human intelligence needed to preprocess even very large "messy" datasets. As opposed to an analyst engaged in manual preprocessing, crowdsourcing offers the possibility to leverage multiple sources of human intelligence, in parallel, thereby improving reliability and speed. Amazon Mechanical Turk (MTurk) stands out as one of the largest crowdsourcing platforms currently available and, its powerful API is now accessible directly in R through **MTurkR**.

MTurk core concepts

Amazon Mechanical Turk is a crowdsourcing platform designed by Amazon as part of its suite of Amazon Web Service (AWS) tools to provide human intelligence for tasks that cannot be readily, affordably, or feasibly automated (Amazon.com, 2012). Because MTurk provides the web application for recruiting, paying, and managing human workers, the effort necessary to move a data cleaning task into the cloud is relatively effortless and, with **MTurkR**, can, in large part, be managed directly in

R. While many early adopters of MTurk as a data generation tool have come from computer science (Mason and Suri, 2012; Kittur et al., 2008), more recent attention has also emerged in the social sciences where MTurk's pool of workers are seen as a low-cost participant pool for human subjects research (Buhrmester et al., 2011; Berinsky et al., 2010; Paolacci et al., 2010). This article provides a sufficiently general overview of MTurk and **MTurkR** to enable its use for a variety of purposes, but focuses primarily on the uses of MTurk for data preprocessing.[1]

Key terms

MTurk connects *requesters*, who are willing to pay *workers* to perform a given task or set of tasks at a specified price per task. These "Human Intelligence Tasks" (HITs), are the core element of the MTurk platform. A HIT is a task that a requester would like one or more workers to perform. Every HIT is automatically assigned a unique HITId to identify this HIT in the system. Performance of that HIT by one worker is called an *assignment*, indexed by a unique AssignmentId, such that a given worker can only complete one assignment per HIT but multiple workers can each complete an assignment for each HIT. As a simple example, if a HIT is a PDF file to be transcribed, the researcher might want three workers to complete the transcription in order to validate the effort and therefore make three assignments available for this HIT.

In other situations, however, a researcher may want workers to complete a set of related tasks. For example, the researcher may want to categorize 5000 text statements such as free response answers to a survey question into a set of fixed categories. Each of these statements could be treated as a separate HIT, grouped as a *HITType* with one (or more) assignment(s) available for each HIT. While a worker could complete all 5000 assignments they might also code fewer (e.g., 50 statements), thereby leaving 4950 assignments for other workers to claim.

Workers choose which HITs to complete and how many HITs they want to complete at any given time, depending on their own time, interests, and the payments that requesters offer in exchange for completing an assignment for a given HIT.[2] A requester can offer as low as $0.005 per assignment. Similarly, requesters can pay any higher amount, but that may not be cost-effective given the market forces in play on MTurk. Workers increasingly expect competitive wages, at a rate of at least U.S. minimum hourly wage.

Once a worker completes a HIT, the requester can *review* the assignment – that is, see the responses provided by the worker to the HIT – and the requester can either *approve* (and thus pay the worker the pre-agreed "reward" amount) or *reject* (and not pay the worker).[3] This review process can be relatively automated or handled manually by the requester.

The MTurk system records all workers that have ever performed work for a given requester and provides an array of functionality for tracking, organizing, paying, and corresponding with workers. In particular, the system allows requesters to regulate who can complete HITs through the use of *QualificationRequirements* (e.g., a worker's previous HIT approval rate, their country of residence, or a requester-defined qualification such as past performance or previously evaluated skills).

Sandbox environment

One final point is that MTurk has both a "live" website and development *sandbox*, where the service can be tested without transacting any money. The sandbox can be a useful place to create and test HITs before making them available for workers. Note, however, that the two systems – despite operating with identical code – have separate databases of HITs, HITTypes, qualifications, workers, and assignments so code may not directly translate between sandbox and the live server.

MTurk API and other packages

Amazon provides software development kits for Python, Ruby, etc. as well as a rudimentary command-line utility, but no officially supported client for R. The **MTurkR** package fills this gap, enabling R

[1] Users specifically interested in social science survey and experimental applications should consult Leeper (2013) and the **MTurkR** documentation.

[2] Workers also communicate about the quality of HITs and requesters on fora such as TurkOpticon (http://turkopticon.differenceengines.com/), MTurk Forum (http://mturkforum.com/), Turker Nation (http://www.turkernation.com/), and Reddit pages (http://www.reddit.com/r/HITsWorthTurkingFor/ and http://www.reddit.com/r/mturk).

[3] Note that Amazon also charges a surcharge on all worker payments. Also, if the requester thinks the work merits additional compensation (or perhaps if workers are rewarded for completing multiple HITs of a given HITType), the requester can also pay a *bonus* of any amount to the worker at any point in the future.

users to fully manage an MTurk workflow, from submitting "messy" data to MTurk, reviewing work completed by workers, and retrieving completed work as an R data frame.[4]

The MTurkR package

Before using MTurk, a MTurk requester account is necessary. These which can be created at http://www.mturk.com.[5] It is also helpful from a practical perspective to have a worker account, so that you can test your own HITs interactively and have the requester-worker relationship necessary to test some MTurk features (e.g., contacting workers or setting up qualifications). **MTurkR**'s access to the MTurk API requires Amazon Access Keys, which can be setup at https://console.aws.amazon.com/iam/home?#security_credential. The *keypair* is a linked *Access Key ID* and a *Secret Access Key*.

MTurkR is implemented in a functional programming style, with the core functionality enabling the creation of HITs and retrieval of resulting assignment data. All of this functionality is described here, as well as in detailed examples in the **MTurkR** package documentation (Leeper, 2016). As a web API client, the package provides a complete wrapper for all API features using function names closely mapped onto API endpoints, making it easy to cross-reference MTurk API documentation with **MTurkR** functionality. **MTurkR** performs HTTP requests to the MTurk API using **curl** (Ooms, 2016) and parses responses using **XML** (Temple Lang, 2012). In almost all cases, responses are converted into data frames. In the event an API request fails, error reporting information is returned instead of the standard data structure.[6]

A simple "hello world!" test in **MTurkR** can be performed by checking the balance in a requester's account. To do so, the AWS credentials are set as environment variables:

```
Sys.setenv("AWS_ACCESS_KEY_ID" = "AWSAccessKeyId")
Sys.setenv("AWS_SECRET_ACCESS_KEY" = "AWSSecretAccessKey")

# Test connection to live server
AccountBalance()

# Test connection to sandbox server
AccountBalance(sandbox = TRUE)
```

`AccountBalance()` returns the current balance in U.S. Dollars; for the sandbox, this is always $10,000. The `sandbox` parameter can also be changed globally with `options("MTurkR.sandbox" = TRUE)`.

Data preprocessing with MTurkR

A common workflow for using MTurk involves starting with a messy data structure and wanting some better-structured resulting data structure (within R this is presumably a data frame). To use **MTurkR**, the analyst must break down the messy data structure into a set of individual tasks (HITs), create those HITs via **MTurkR**, allow time for workers to complete assignments, and then collect and review completed assignments before proceeding with the analysis of the resulting data in R. How to achieve this in **MTurkR**? I begin by demonstrating how to create a single HIT and then demonstrating more convenient wrapper functions for creating batches of HITs in bulk.

Creating individual HITs

First, creating a HIT requires registering a HITType, which sets various worker-visible characteristics of the HIT(s), four of which are required and three that are optional:

- Title, short title for the HIT to be displayed to workers (required).
- Description, a description of the HIT to be displayed to workers (required).
- Reward, in U.S. Dollars (required).

[4] **MTurkR** also offers a set of interactive command-line menus for performing **MTurkR** operations without the need to write any code. An add-on package called **MTurkRGUI** (Leeper, 2015) implements an even more robust graphical user interface using the cross-platform **tcltk** package. Additional details about these **MTurkR** features are available in the package documentation and on the **MTurkR** wiki at https://www.github.com/leeper/MTurkR.

[5] Note that MTurk is currently only available to requesters with a United States address and a Social Security number.

[6] As a convenience, by default, all API requests and responses are stored in a tab-separated-value log file in the user's working directory, alongside information about API requests.

- Duration, in seconds (required).
- Keywords, a comma-separated list of keywords used by workers to search for HITs (optional; default is empty).
- Assignment auto-approval delay, a time in seconds which specifies when assignments will automatically be paid if not first rejected (optional; default is 30 days).
- Qualification requirements, a complex structure which controls which workers can complete the HIT (optional; default is none).

To register a HITType, at least the first four characteristics just described need to be defined in a call to `RegisterHITType()`, for example:

```
hittype1 <- RegisterHITType(title = "Tell us something",
                            description = "Answer a single question",
                            reward = "0.05",
                            duration = seconds(days = 1, hours = 8),
                            keywords = "text, answer, question",
                            auto.approval.delay = seconds(days = 1))
```

MTurkR's `seconds()` function provides a convenient way of converting time measurements in days, hours, minutes, or seconds into a total number of seconds. With the HITType created, one can begin creating individual HITs associated with that HITType using `CreateHIT()`.

A HIT consists of a HITType and various HIT-specific attributes, the most import of which is a "question" text specifying the contents of the task as shown to the worker via an HTML iframe on the MTurk worker website. Questions can be specified in one of several ways:

- An HTTPS URL (or "ExternalQuestion") for a page containing the HIT HTML.
- An "HTMLQuestion" structure, essentially the HTML to display to the worker.
- A "QuestionForm" structure, which is a proprietary markup language used by MTurk.
- A "HITLayoutID" value retrieved from the MTurk requester website[7].

In addition to one of the above question specifications, the other HIT attributes are:

- Duration, the number of assignments to be created for the HIT (required; default 1).
- Expiration, a time specifying when the HIT will expire and thus be unavailable to workers, in seconds (required; no default).
- Annotation, specifying a hidden value that describes the HIT as a reference for the requester (optional; default is empty).

In most cases, specifying an HTMLQuestion is the easiest approach. This simply means writing a complete, HTML5-compliant document creating a web form that will display some material to the worker and allow them to enter and submit answer information to the server. Some examples are installed with **MTurkR**, such as:

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv='Content-Type' content='text/html; charset=UTF-8'/>
  <script type='text/javascript'
   src='https://s3.amazonaws.com/mturk-public/externalHIT_v1.js'></script>
 </head>
 <body>
  <form name='mturk_form' method='post' id='mturk_form'
   action='https://www.mturk.com/mturk/externalSubmit'>
  <input type='hidden' value='' name='assignmentId' id='assignmentId'/>
  <h1>What's up?</h1>
  <p><textarea name='comment' cols='80' rows='3'></textarea></p>
  <p><input type='submit' id='submitButton' value='Submit' /></p></form>
  <script language='Javascript'>turkSetAssignmentID();</script>
 </body>
</html>
```

[7] This is useful for creating HITs using **MTurkR** based on templates created on the MTurk requester website.

Workers will see a rendered version of the HTMLQuestion, specifically a question – "What's up?" – and a multi-line text response they can complete. The JavaScript in the HTMLQuestion is essential for the HIT to behave properly. To setup this HIT in the MTurk system, use `CreateHIT()` passing it the HITTypeId created earlier, making the HIT available for 4 days and setting a private annotation field to remind us about the HIT:

```
f1 <- system.file("templates/htmlquestion1.xml", package = "MTurkR")
hq <- GenerateHTMLQuestion(file = f1)
hit1 <- CreateHIT(hit.type = hittype1$HITTypeId,
                  question = hq$string,
                  expiration = seconds(days = 4),
                  annotation = "my first HIT")
```

At this point, a worker needs to submit the assignment. Once that has happened (this can be checked using `HITStatus()` or `GetHIT(hit = hit$HITId)`), the assignment data can be retrieved through:

```
# Retrieve all assignments for a HIT
a1 <- GetAssignments(hit = hit1$HITId)

# Retrieve all assignments for all HITs for a HITType
a2 <- GetAssignments(hit.type = hittype1$HITTypeId)

# Retrieve a specific assignment
a3 <- GetAssignments(assign = a1$AssignmentId[1])
```

These assignments will be automatically approved after one day (according to the value specified in `auto.approval.delay` when registering the HITType). Assignments can be approved manually using `ApproveAssignment()`:

```
# Approve 1 assignment
ApproveAssignments(assignments = a1$AssignmentId[1],
                   feedback = "Well done!")

# Approve multiple assignments
ApproveAssignments(assignments = a1$AssignmentId)

# Approve all assignments for a HIT
ApproveAllAssignments(hit = hit1$HITId)

# Approve all assignments for all HITs of a HITType
ApproveAllAssignments(hit = hittype1$HITTypeId)

# Approve all assignments based on annotation
ApproveAllAssignments(annotation = "my first HIT")
```

Rejecting HITs works identically to the above but using `RejectAssignments()`. Feedback is optional for assignment approval but required for assignment rejection.[8] Feedback is passed through the `feedback` argument.

Managing crowdworkers with QualificationTypes

One important consideration when creating a HIT is that, by default, every HIT is available to all MTurk workers unless QualificationRequirements have been specified in the `RegisterHITType()` operation. Furthermore, these QualificationRequirements are attached to a HITType, not an individual HIT, so HITs directed at distinct subsets of workers need to be attached to distinct HITTypes.

There are several built-in QualificationTypes that can be used as QualificactionRequirements, including country of residence and various measures of experience on MTurk (e.g., number of HITs completed, approval rate, etc.). To configure a HITType that will only be available to workers in the United States who have completed more than 500 approved HITs, first use `GenerateQualificationRequirement()` to setup a QualificationRequirement structure locally. This involves naming the QualificationTypes to use in the QualificationRequirement, along with "comparators" and "values", which are interpreted as logical statements of the form "Locale is equal to US" and "NumberApproved is greater than 500":

[8] Rejected assignments can also be converted to approved within 30 days of rejection, though the reverse operation is not possible.

```
# Shorthand names of location and approval qualifications
q_names <- c("Locale", "NumberApproved")

# Comparators ("==" for location and ">" for past approvals)
q_comparators <- c("==", ">")

# Qualification values ("US" for location and "500" for past approvals)
q_values <- c("US", 500)

# Convert these values into a QualificationRequirement
qreq2 <- GenerateQualificationRequirement(q_names,
                                          q_comparators,
                                          q_values,
                                          preview = TRUE)
```

This structure is passed as the `qual.req` argument to `RegisterHITType()` to create a new HITType with these QualificationRequirements:

```
# Register HITType using the QualificationRequirement
hittype2 <- RegisterHITType(title = "Tell us something",
                            description = "Answer a single question",
                            reward = "0.05",
                            duration = seconds(days = 1, hours = 8),
                            keywords = "text, answer, question",
                            auto.approval.delay = seconds(days = 15),
                            qual.req = qreq2)
```

This attaches a QualificationRequirement to all HITs created within this new HITType, preventing workers who fail to meet the qualifications from working on them (or in this case, given `preview = TRUE`, even viewing the HITs).[9]

In addition to using the built-in QualificationTypes, workers can also be managed in other ways. One way is to block workers who consistently perform inadequate work using `BlockWorkers()`. This should be used sparingly, however, as workers who are repeatedly blocked will have their MTurk accounts disabled. A data frame of previously blocked workers is return by `GetBlockedWorkers()`. `UnblockWorkers()` is provided to unblock workers. In addition, it is possible to email workers using `ContactWorkers()` and supply optional bonus payments using `GrantBonus()`. These can be useful for managing complex projects, incentivizing good work, and inviting well-performing workers to complete new projects.

QualificationRequirements set for a HITType can also be used to manage workers' access to HITs. The built-in QualificationTypes are quite useful for this, but requesters can also create more tailored QualificationTypes based on other criteria. A common use case is to only allow new workers to complete a HIT. The steps to achieve this are: create a new QualificationType, assign different values for that QualificationType to past and new workers, and then create a new HITType using this QualificationType as a QualificationRequirement.

```
# Create the QualificationType
thenewqual <- CreateQualificationType(name = "Prevent Retakes",
                                      description = "Worked for me before",
                                      status = "Active",
                                      auto = TRUE,
                                      auto.value = 100)

# Assign qualification
AssignQualification(qual = thenewqual$QualificationTypeId,
                    workers = hit1$WorkerId,
                    value = "50")

# Generate QualificationRequirement
qreq3 <-
  GenerateQualificationRequirement(thenewqual$QualificationTypeId, "==", "100")
```

[9] HITTypes cannot be edited. If you attempt to create two HITTypes with identical properties, they will be assigned the same HITTypeId. If you modify any attribute, a new HITType will be created. If you have HITs that you would like to assign to a different HITType, use `ChangeHITType()`.

```
# Create HIT, implicitly generating HITType
hit2 <- CreateHIT(question = hq$string,
                  expiration = seconds(days = 4),
                  assignments = 10,
                  title = "Tell us something",
                  description = "Answer a single question",
                  reward = "0.05",
                  duration = seconds(days = 1, hours = 8),
                  keywords = "text, answer, question",
                  auto.approval.delay = seconds(days = 15),
                  qual.req = qreq3,
                  annotation = "my second HIT")
```

To explain what is happening here, a new QualificationType was created that workers can "request" through the MTurk website. If they request it, they will automatically be assigned a score of 100 on the QualificationType. This QualificationType was assigned to all of our workers from the first HIT but at a score lower than the automatically granted value. Next, a QualificationRequirement was created that makes a HIT only available to those with the automatically granted value, and, finally, this was attached to a HITType that is created automatically within the call to `CreateHIT()`. Now 10 new workers can complete this HIT, excluding the worker(s) that completed work on the first HIT.

QualificationTypes and QualificationRequirements on HITTypes allow a requester to manage a large pool of workers in complex ways. Workers that have been assigned scores on a QualificationType can be retrieved using `GetQualifications()`, or modified using `UpdateQualificationScore()`. The attributes of the QualificationType itself can be changed using `UpdateQualificationType()`, and the QualificationType and all associated scores can be deleted using `DisposeQualificationTypes()`.[10] QualificationTypes can also be configured with a "qualification test" that allows workers to submit provisional work as a measure of abilities and then qualifications can be approved/revoked manually based on their responses or even configured with an "AnswerKey" that will automatically evaluate the worker's test performance and assign a score for the QualificationType. Again, the **MTurkR** documentation includes extended examples and possible use cases.

When finished with a HIT and all of its assignment data, it can be deleted from the system using `DisposeHIT()`. This is not a reversible action, so it should be used with caution. HITs will be deleted automatically by Amazon after a period of inactivity, but cleaning up unneeded HITs can be useful given that there is no particularly good way to search for HITs within the system. The `SearchHITs()` operation simply returns a sorted data frame of all HITs.

Creating multiple HITs

In addition to creating single HITs, **MTurkR** offers functionality to manage very large projects involving many HITs. This section describes that functionality in detail.

There are four functions that have been added to **MTurkR** as of v0.6.5 (available on CRAN since 25 May 2015) to facilitate the bulk creation of HITs, for example for the earlier use case of creating a training set of open-ended text responses for a classification algorithm. These functions are wrappers for `CreateHIT()` designed to accept different kinds of input for the `question` argument and cycle through those inputs to create multiple HITs. They are:

- `BulkCreate()` provides a low-level loop around `CreateHIT()` that takes a character vector of question values as input.
- `BulkCreateFromHITLayout()` provides functionality for creating multiple HITs from a HITLayout created on the MTurk Requester website.
- `BulkCreateFromTemplate()` provides higher-level functionality that translates a HIT template and a data frame of input values into a series of HITs.
- `BulkCreateFromURLs()` provides a convenient way of creating multiple HITs from a character vector of URLs.

The last two of these are likely to be the most useful, so extended examples are provided below.

`GenerateHITsFromTemplate()` works from a template HTMLQuestion document containing placeholders for input values and a data frame of values, one set of values per row. An example template is installed with **MTurkR**:

[10] If a QualificationType is requestable but not automatically approved, qualification scores have to be granted manually by the requester. The additional functions `GetQualificationRequests()`, `GrantQualification()`, and `RevokeQualification()` can be used to manage requests.

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv='Content-Type' content='text/html; charset=UTF-8'/>
  <script type='text/javascript'
   src='https://s3.amazonaws.com/mturk-public/externalHIT_v1.js'></script>
 </head>
 <body>
  <form name='mturk_form' method='post' id='mturk_form'
   action='https://www.mturk.com/mturk/externalSubmit'>
  <input type='hidden' value='' name='assignmentId' id='assignmentId'/>
  <h1>${hittitle}</h1>
  <p>${hitvariable}</p>
  <p>What do you think?</p>
  <p><textarea name='comment' cols='80' rows='3'></textarea></p>
  <p><input type='submit' id='submitButton' value='Submit' /></p></form>
  <script language='Javascript'>turkSetAssignmentID();</script>
 </body>
</html>
```

This template contains two placeholders '`${hittitle}`' and '`${hitvariable}`'. These placeholders will by replaced by `GenerateHITsFromTemplate()` with values specified by the `hittitle` and `hitvariable` columns in an input data frame, creating a set of unique HITs as one batch.

```
# Create input data frame
inputdf <- data.frame(hittitle = c("HIT title 1", "HIT title 2", "HIT title 3"),
                      hitvariable = c("HIT text 1", "HIT text 2", "HIT text 3"),
                      stringsAsFactors = FALSE)

# Create HITs
bulk1 <-
  BulkCreateFromTemplate(template = system.file("template.html", package = "MTurkR"),
                         input = inputdf,
                         annotation = paste("Bulk From Template", Sys.Date()),
                         title = "Describe a text",
                         description = "Describe this text",
                         reward = ".05",
                         expiration = seconds(days = 4),
                         duration = seconds(minutes = 5),
                         auto.approval.delay = seconds(days = 1),
                         keywords = "categorization, image, moderation, category")
```

The response structure for these functions is a list of single-row data frames. If all HIT creation operations succeed, then the response can easily be converted using `do.call("rbind",bulk2)` to a data frame, but users will typically only need to examine this structure if errors occurred. Details about the individual HITs can be retrieved at any time using `GetHITs()` or `SearchHITs()`.

At this point workers need to complete their assignments. Because the same value for the `annotation` was supplied to all of these HITs, the results for all associated assignments can easily be retrieved using `GetAssignments()`:

```
# Get assignments using annotation
a1 <- GetAssignments(annotation = paste("Bulk From Template", Sys.Date()))
# Get assignments using HITTypeId
a2 <- GetAssignments(hit.type = bulk1[[1]]$HITTypeId)
```

Unfortunately, MTurk does not return the contents of the `question` parameter with the completed assignments. However HITId is included so it is trivial to merge the input data frame with the assignment data frame allowing the comparison of the original data (e.g., open-ended response text) to the information supplied by workers (e.g., the classification):

```
# Extract HITIds from `bulk1`
inputvalues$HITId <- do.call("rbind", bulk1)$HITId

# Merge `inputvalues` and `assignmentresults`
merge(inputdf, a1, all = TRUE, by = "HITId")
```

`BulkCreateFromURLs()` behaves similarly but accepts a character vector of URLs to be used as ExternalQuestion values. This function requires a `frame.height` argument to specify the vertical size of the HIT as shown to workers.[11]

```
bulk2 <-
  BulkCreateFromURLs(url = paste0("https://www.example.com/", 1:3, ".html"),
                     frame.height = 450,
                     annotation = paste("Bulk From URLs", Sys.Date()),
                     title = "Categorize an image",
                     description = "Categorize this image",
                     reward = ".05",
                     expiration = seconds(days = 4),
                     duration = seconds(minutes = 5),
                     auto.approval.delay = seconds(days = 1),
                     keywords = "categorization, image, moderation, category")
```

Addressing problems

Sometimes things go wrong. Perhaps the HITs contained incorrect information or the work being performed is of low quality because of a mistake in the HIT's instructions. When these situations occur, it is easy to address problems using a host of HIT-management functions. To expire a HIT early, simply call `ExpireHIT()` specifying a HITId, HITTypeId, or annotation value. To delay the expiration of HIT by a specified number of seconds use `ExtendHIT()` with its `add.seconds` argument. A call to `ExtendHIT()` with the `add.assignments` parameter increases the number of available assignments for the HIT(s).[12]

One other useful set of operations provided by MTurk is a "notification" system that allows requesters to receive messages about various HITType events either via email or to an AWS Simple Queue Service (SQS) queue (see **MTurkR** documentation for examples of the latter). Notifications can be triggered by various events and can be used as an alternative to actively monitoring the status of a HIT vai `HITStatus()`. Here is an example notification to send an email whenever a HIT of a given HITType expires:

```
n <- GenerateNotification("requester@example.com",
                          event.type = "HITExpired")
SetHITTypeNotification(hit.type = hittype1$HITTypeId,
                       notification = n,
                       active = TRUE)
```

An example of massive-scale photo rating

To demonstrate the ease with which **MTurkR** can be used to preprocess a massive amount of data, I provide an example of a large-scale photo-rating task. Here, I was interested in obtaining a rating of "facial competence" for U.S. politicians compared with ratings of faces from the general U.S. population. Facial competence is said to enhance politicians' electoral success, but previous studies have never compared these to a general population sample. Are politicians generally more facially competent than other individuals? While this is a modest research question, it demonstrates well the immense human effort needed to draw even simple conclusions from messy data structures.

To provide a sampling of politicians' faces, I scraped photos of 533 members of the 113th U.S. Congress from the website of the Government Printing Office. I then combined these photo data with 5000 randomly sampled images from the 10K U.S. Adult Faces Database (Bainbridge et al., 2013), which provides a nationally representative sampling of U.S. faces, and standardized the image size and resolution across all faces.[13] To rate facial competence, I created a simple one-question HIT using HTML (see Figure 1) that displayed one of the faces and asked for a rating of facial competence on a 0 to 10 scale.[14] I include the complete HTML file in the supplemental material for this article.

[11] MTurk displays the page specified by the ExternalQuestion URL inside an HTML iframe on the worker site.

[12] Note that this number must be positive and, therefore, the number of available assignments cannot be reduced. If it is needed to reduce the number of assignments completed for a HIT, the HIT can be expired once the desired number of assignments have been completed.

[13] Complete code to perform the scraping and image processing are provided along with supplemental material for this article at https://github.com/leeper/mturkr-article (http://dx.doi.org/10.5281/zenodo.33595).

[14] The HIT additionally included questions to address possible problems (i.e., a subject recognizes a face or the image did not display properly).

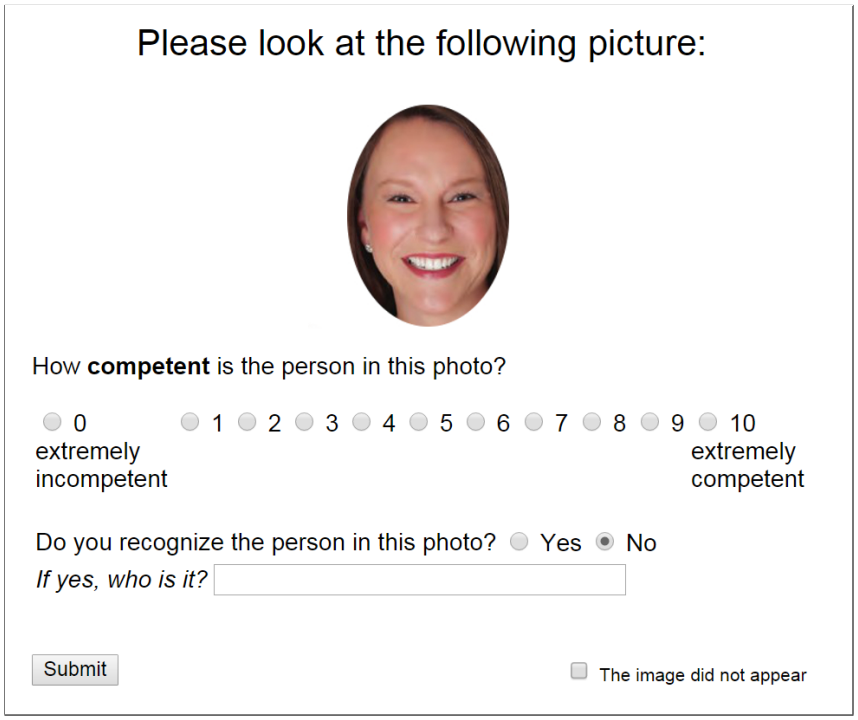


Figure 1: Example photo rating HIT.

After uploading all 5533 images to an Amazon Simple Storage Service (S3) bucket, which is a simple cloud storage facility, to make the files publicly available[15] and storing their filenames in a local RDS file, it was trivial to send these images to MTurk workers for categorization. To ensure reliability of the results, each face was rated by 5 workers. Workers were given 45 seconds to rate each face and were paid $0.01 per face. The 27,665 images were rated by a team of 225 U.S.-based workers over a period of 75 minutes. The entire operation cost $412.50. Achieving this required three steps in **MTurkR**: (1) creating a QualificationRequirement to restrict the task to U.S.-based workers with 95% approval ratings, (2) registering a HITType into which the HITs will be created, and (3) the creation of a batch of HITs using `BatchCreateFromURLs()`.

```
# Setup QualificationRequirement
## U.S.-based, 95% approval on HITs
qual <-  GenerateQualificationRequirement(c("Locale", "Approved"),
                                          c("==", ">"),
                                          c("US", 95),
                                          preview = TRUE)

# Register HITType
desc <- "Judge the competence of a person from an image of their face.
  The HIT involves only one question: a rating of the competence of the
  person. You have 45 seconds to complete the HIT. There are several
  thousand HITs available in this batch. If you recognize the person,
  please enter their name in the space provided; your work will still be
  approved even if you recognize the face."
hittype <-
  RegisterHITType(title = "Rate the competence of a person",
                  description = desc,
                  reward = "0.01",
                  duration = seconds(seconds = 45),
                  auto.approval.delay = seconds(days = 1),
                  qual.req = qual,
                  keywords = "categorization, photo, image, rating, fast, easy")

# All faces were loaded into Amazon S3
```

[15] Any public file host could be used, not just S3.

```
s3url <- "https://s3.amazonaws.com/mturkfaces/"
# File names were saved as a character vector locally
faces <- readRDS("faces_all.RDS")
d <- data.frame(face = paste0(s3url,faces),
                stringsAsFactors = FALSE)

# Create 5500 HITs
bulk <- BulkCreateFromTemplate(template = "mturk.html",
                               frame.height = 550,
                               input = d,
                               hit.type = hittype$HITTypeId,
                               expiration = seconds(days = 7),
                               # 5 assignments/face
                               assignments = 5,
                               annotation = "Face Categorization 2015-06-08")
```

Using the specified `annotation` value, `GetAssignments()` returns a large data frame with 27670 rows and 25 columns:

```
a <- GetAssignments(annotation = "Face Categorization 2015-06-08")
dim(a)
# [1] 27670    25
names(a)
#  [1] "AssignmentId"          "WorkerId"              "HITId"
#  [4] "AssignmentStatus"      "AutoApprovalTime"      "AcceptTime"
#  [7] "SubmitTime"            "ApprovalTime"          "RejectionTime"
# [10] "RequesterFeedback"     "ApprovalRejectionTime" "SecondsOnHIT"
# [13] "competent"             "recognized"            "name"
# [16] "face"                  "condition"             "browser"
# [19] "engine"                "platform"              "language"
# [22] "width"                 "height"                "resolution"
# [25] "problem"
```

Most of the columns contain metadata for identifying each assignment (AssignmentId, WorkerId, HITId), metadata about the completion of the assignment (AssignmentStatus, AutoAprpovalTime, AcceptTime, SubmitTime, ApprovalTime, RejectionTime, RequesterFeedback, ApprovalRejectionTime, SecondsOnHIT), and then several columns displaying responses to the three HIT questions displayed to the workers: `competent`, `recognized`, and `name`. The names of these variables are given by the `name` attribute of the radio buttons used in the HTMLQuestion form. The data frame also contains additional variables that record metadata about the worker's browser, which were recorded automatically via Javascript.

As noted earlier, a limitation of the MTurk API is that it does not return information about the values of variables replaced in the templating process, so it can be difficult to identify which assignment(s) correspond to which input values. To circumvent this limitation, this HIT template was designed to use the `${face}` variable twice: once to actually display the image to the worker and once to record its value in a hidden field called `face` in the HTMLQuestion form. As a result, this variable becomes available to us in the results data frame.

Setup in this way, it becomes trivial to analyze facial competence ratings of politicians and those from the general population sample. To perform the analysis, I simply conducted a Mann-Whitney-Wilcoxon test for a difference in competence ratings between the faces of politicians and non-politicians. (In these data, politicians' photos were identified by a simple pattern matching file name. This would have more easily been done with a hidden HTML variable when creating the batch.) So, I extract the two variables from the assignment data frame, convert them to numeric, and perform the test:

```
competence <- as.numeric(a$competent)
politician <- as.numeric(grepl("[[:digit:]]{2}-[[:digit:]]{3}", a$face))

round(prop.table(table(politician, competence), 1), 2)
#
#           competence
# politician    0    1    2    3    4    5    6    7    8    9   10
#          0 0.03 0.03 0.04 0.07 0.11 0.13 0.17 0.17 0.16 0.06 0.02
#          1 0.01 0.01 0.02 0.04 0.07 0.12 0.19 0.20 0.22 0.09 0.04

wilcox.test(competence ~ politician)
```

```
#
#         Wilcoxon rank sum test with continuity correction
#
# data:  competence by politician
# W = 26886000, p-value < 2.2e-16
# alternative hypothesis: true location shift is not equal to 0
```

Politicians do appear to have higher facial competence. While this is a fairly trivial analytic conclusion, it demonstrates the ease with which crowdsourced human intelligence can be leveraged to preprocess a massive amount of data, translating messy sources into easily analyzed data. Because crowdsourcing is inherently massively parallel, it dramatically reduces the amount of time needed to parse a rough data source. In this case, the MTurk workers created the completed dataset in about 75 minutes. Were a single individual to attempt this task alone and it took (as a generous estimate) only 5 seconds to categorize each face, the task would be completed in 38.4 hours, or about 31-times as long as with MTurk.

Conclusion

This paper has described the MTurk platform and offered an introduction to the R package **MTurkR** focused on preprocessing of messy data for immediate use in R. In short, **MTurkR** provides a stable, well-developed R interface to one of the largest crowdsourcing sites presently available. The package has been developed and refined for more than three years, has extensive in-package and online documentation, and is incredibly easy to use. By providing a low-level wrapper to the Amazon Mechanical Turk API, it also means that **MTurkR** could serve well as the basis for much more sophisticated R applications that leverage human intelligence as an enhancement to the computational features already available in R.

Bibliography

Amazon.com. Amazon Mechanical Turk getting started guide, 2012. URL http://docs.amazonwebservices.com/AWSMechTurk/latest/AWSMechanicalTurkGettingStartedGuide/Welcome.html?r=4925. [p276]

W. A. Bainbridge, P. Isola, and A. Oliva. The instrinsic memorability of face photographs. *Journal of Experimental Psychology: General*, 142(4):1323–1334, 2013. doi: 10.1037/a0033872. [p284]

A. J. Berinsky, G. A. Huber, and G. S. Lenz. Using Mechanical Turk as a subject recruitment tool for experimental research. Unpublished paper, 2010. [p277]

M. Buhrmester, T. Kwang, and S. D. Gosling. Amazon's Mechanical Turk: A new source of inexpensive, yet high-quality, data? *Perspectives on Psychological Science*, 6(1):3–5, Feb. 2011. doi: 10.1177/1745691610393980. [p277]

J. J. Chen, N. J. Menezes, and A. D. Bradley. Opportunities for crowdsourcing research on Amazon Mechanical Turk. Unpublished paper, 2011. [p276]

A. Kittur, E. H. Chi, and B. Suh. Crowdsourcing user studies with Mechanical Turk. In *CHI 2008 – Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, page 453, New York, New York, USA, 2008. ACM Press. doi: 10.1145/1357054.1357127. [p277]

T. J. Leeper. Crowdsourcing with R and the MTurk API. *The Political Methodologist*, 20(2):2–7, 2013. [p277]

T. J. Leeper. *MTurkRGUI: A Graphical User Interface for MTurkR*, 2015. URL https://CRAN.R-project.org/package=MTurkRGUI. R package version 0.1.5. [p278]

T. J. Leeper. *MTurkR: R Client for the MTurk Requester API*, 2016. URL https://www.github.com/leeper/MTurkR. R package version 0.7.0. [p276, 278]

W. Mason and S. Suri. Conducting behavioral research on Amazon's Mechanical Turk. *Behavior Research Methods*, 44(1):1–23, Mar. 2012. doi: 10.3758/s13428-011-0124-6. [p277]

J. Ooms. *curl: A Modern and Flexible Web Client for R*, 2016. URL https://CRAN.R-project.org/package=curl. R package version 0.9.6. [p278]

G. Paolacci, J. Chandler, and L. N. Stern. Running experiments on Amazon Mechanical Turk. *Judgment and Decision Making*, 5(5):411–419, 2010. [p277]

L. A. Schmidt. Crowdsourcing for human subjects research. In *CrowdConf 2010*, San Francisco, CA, 2010. [p276]

D. Temple Lang. *XML: Tools for Parsing and Generating XML within R and S-Plus*, 2012. URL http://CRAN.R-project.org/package=XML. R package version 3.9-4.1. [p278]

Thomas J. Leeper
Department of Government
London School of Economics and Political Science
London, United Kingdom
thosjleeper@gmail.com

mclust 5: Clustering, Classification and Density Estimation Using Gaussian Finite Mixture Models

by Luca Scrucca, Michael Fop, T. Brendan Murphy and Adrian E. Raftery

Abstract Finite mixture models are being used increasingly to model a wide variety of random phenomena for clustering, classification and density estimation. **mclust** is a powerful and popular package which allows modelling of data as a Gaussian finite mixture with different covariance structures and different numbers of mixture components, for a variety of purposes of analysis. Recently, version 5 of the package has been made available on CRAN. This updated version adds new covariance structures, dimension reduction capabilities for visualisation, model selection criteria, initialisation strategies for the EM algorithm, and bootstrap-based inference, making it a full-featured R package for data analysis via finite mixture modelling.

Introduction

mclust (Fraley et al., 2016) is a popular R package for model-based clustering, classification, and density estimation based on finite Gaussian mixture modelling. An integrated approach to finite mixture models is provided, with functions that combine model-based hierarchical clustering, EM for mixture estimation and several tools for model selection. Thus **mclust** provides a comprehensive strategy for clustering, density estimation and discriminant analysis. A variety of covariance structures obtained through eigenvalue decomposition are available. Functions for performing single E and M steps and for simulating data for each available model are also included. Additional ways of displaying and visualising fitted models along with clustering, classification, and density estimation results are also provided. It has been used in a broad range of contexts including geochemistry (Templ et al., 2008; Ellefsen et al., 2014), chemometrics (Fraley and Raftery, 2006a, 2007b), DNA sequence analysis (Verbist et al., 2015), gene expression data (Yeung et al., 2001; Li et al., 2005; Fraley and Raftery, 2006b), hydrology (Kim et al., 2014), wind energy (Kazor and Hering, 2015), industrial engineering (Campbell et al., 1999), epidemiology (Flynt and Daepp, 2015), food science (Kozak and Scaman, 2008), clinical psychology (Suveg et al., 2014), political science (Ahlquist and Breunig, 2012; Jang and Hitchcock, 2012), and anthropology (Konigsberg et al., 2009).

One measure of the popularity of **mclust** is provided by the download logs of the RStudio (`http://www.rstudio.com`) CRAN mirror (available at `http://cran-logs.rstudio.com`). The **cranlogs** package (Csardi, 2015) makes it easy to download such logs and graph the number of downloads over time. We used **cranlogs** to query the RStudio download database over the past three years. In addition to **mclust**, other R packages which handle Gaussian finite mixture modelling as part of their capabilities have been included in the comparison: **Rmixmod** (Lebret et al., 2015), **mixture** (Browne et al., 2015), **EMCluster** (Chen and Maitra, 2015), **mixtools** (Benaglia et al., 2009), and **bgmm** (Biecek et al., 2012). We also included **flexmix** (Leisch, 2004; Grün and Leisch, 2007, 2008) which provides a general framework for finite mixtures of regression models using the EM algorithm, since it can be adapted to perform Gaussian model-based clustering using a limited set of models (only the diagonal and unconstrained covariance matrix models). Table 1 summarises the functionalities of the selected packages.

Package	Version	Clustering	Classification	Density estimation	Non-Gaussian components
mclust	5.2	✓	✓	✓	✗
Rmixmod	2.0.3	✓	✓	✗	✓
mixture	1.4	✓	✓	✗	✗
EMCluster	0.2-5	✓	✓	✗	✗
mixtools	1.0.4	✓	✗	✓	✓
bgmm	1.7	✓	✓	✗	✗
flexmix	2.3-13	✓	✗	✗	✓

Table 1: Capabilities of the selected packages dealing with finite mixture models.

Figure 1 shows the trend in weekly downloads from the RStudio CRAN mirror for the selected

packages. The popularity of **mclust** has been increasing steadily over time with a first high peak around mid April 2015, probably due to the release of R version 3.2 and, shortly after, the release of version 5 of **mclust**. Then, successive peaks occurred in conjunction with the release of package's updates. Based on these logs, **mclust** is the most downloaded package dealing with Gaussian mixture models, followed by **flexmix** which, as mentioned, is a more general package for fitting mixture models but with limited clustering capabilities.

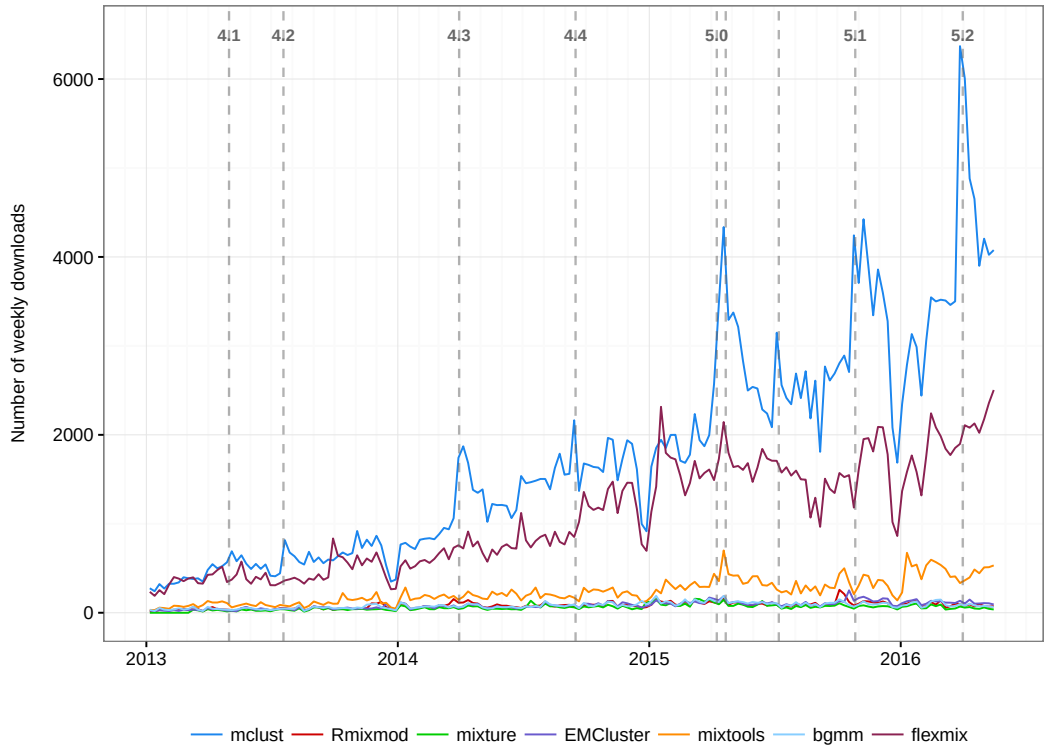


Figure 1: Number of weekly downloads from the RStudio CRAN mirror over time for some R packages dealing with Gaussian finite mixture modelling.

Another aspect that can be considered as a proxy for the popularity of a package is the mutual dependencies structure between R packages[1]. This can be represented as a graph with packages at the vertices and dependencies (either "Depends", "Imports", "LinkingTo", "Suggests" or "Enhances") as directed edges, and analysed through the *PageRank* algorithm used by the Google search engine (Brin and Page, 1998). For the packages considered previously, we used the `page.rank` function available in the **igraph** package (Csardi and Nepusz, 2006) and we obtained the ranking reported in Table 2, which approximately reproduces the results discussed above. Note that **mclust** is among the top 100 packages on CRAN by this ranking. Finally, its popularity is also indicated by the 55 other CRAN packages listed as reverse dependencies, either "Depends", "Imports" or "Suggests".

mclust	Rmixmod	mixture	EMCluster	mixtools	bgmm	flexmix
75	2300	2319	2143	1698	3736	270

Table 2: Ranking obtained with the *PageRank* algorithm for some R packages dealing with Gaussian finite mixture modelling. At the time of writing there are 8663 packages on CRAN.

Earlier versions of the package have been described in Fraley and Raftery (1999), Fraley and Raftery (2003), and Fraley et al. (2012). In this paper we discuss some of the new functionalities available in **mclust** version ≥ 5. In particular we describe the newly available models, dimension reduction for visualisation, bootstrap-based inference, implementation of different model selection criteria and initialisation strategies for the EM algorithm.

The reader should first install the latest version of the package from CRAN with

```
> install.packages("mclust")
```

[1]See http://piccolboni.info/2012/05/essential-r-packages.html.

Then the package is loaded into an R session using the command

```
> library(mclust)
    __  ___________    __  _____________
   /  |/  / ____/ /   / / / / ___/_  __/
  / /|_/ / /   / /   / / / /\__ \ / /
 / /  / / /___/ /___/ /_/ /___/ // /
/_/  /_/\____/_____/\____//____//_/    version 5.2
Type 'citation("mclust")' for citing this R package in publications.
```

All the datasets used in the examples are available in **mclust** or in other R packages, such as **gclus** (Hurley, 2012), **rrcov** (Todorov and Filzmoser, 2009) and **tourr** (Wickham et al., 2011), and can be installed from CRAN using the above procedure, except where noted differently.

Gaussian finite mixture modelling

Let $\boldsymbol{x} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_n\}$ be a sample of n independent identically distributed observations. The distribution of every observation is specified by a probability density function through a finite mixture model of G components, which takes the following form

$$f(\boldsymbol{x}_i; \boldsymbol{\Psi}) = \sum_{k=1}^{G} \pi_k f_k(\boldsymbol{x}_i; \boldsymbol{\theta}_k), \tag{1}$$

where $\boldsymbol{\Psi} = \{\pi_1, \ldots, \pi_{G-1}, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_G\}$ are the parameters of the mixture model, $f_k(\boldsymbol{x}_i; \boldsymbol{\theta}_k)$ is the kth component density for observation $\boldsymbol{x}_i$ with parameter vector $\boldsymbol{\theta}_k$, $(\pi_1, \ldots, \pi_{G-1})$ are the mixing weights or probabilities (such that $\pi_k > 0$, $\sum_{k=1}^{G} \pi_k = 1$), and G is the number of mixture components.

Assuming that G is fixed, the mixture model parameters $\boldsymbol{\Psi}$ are usually unknown and must be estimated. The log-likelihood function corresponding to equation (1) is given by $\ell(\boldsymbol{\Psi}; \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \sum_{i=1}^{n} \log(f(\boldsymbol{x}_i; \boldsymbol{\Psi}))$. Direct maximisation of the log-likelihood function is complicated, so the maximum likelihood estimator (MLE) of a finite mixture model is usually obtained via the EM algorithm (Dempster et al., 1977; McLachlan and Peel, 2000).

In the model-based approach to clustering, each component of a finite mixture density is usually associated with a group or cluster. Most applications assume that all component densities arise from the same parametric distribution family, although this need not be the case in general. A popular model is the Gaussian mixture model (GMM), which assumes a (multivariate) Gaussian distribution for each component, i.e. $f_k(\boldsymbol{x}; \boldsymbol{\theta}_k) \sim N(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$. Thus, clusters are ellipsoidal, centered at the mean vector $\boldsymbol{\mu}_k$, and with other geometric features, such as volume, shape and orientation, determined by the covariance matrix $\boldsymbol{\Sigma}_k$. Parsimonious parameterisations of the covariances matrices can be obtained by means of an eigen-decomposition of the form $\boldsymbol{\Sigma}_k = \lambda_k \boldsymbol{D}_k \boldsymbol{A}_k \boldsymbol{D}_k^\top$, where λ_k is a scalar controlling the volume of the ellipsoid, $\boldsymbol{A}_k$ is a diagonal matrix specifying the shape of the density contours with $\det(\boldsymbol{A}_k) = 1$, and $\boldsymbol{D}_k$ is an orthogonal matrix which determines the orientation of the corresponding ellipsoid (Banfield and Raftery, 1993; Celeux and Govaert, 1995). In one dimension, there are just two models: `E` for equal variance and `V` for varying variance. In the multivariate setting, the volume, shape, and orientation of the covariances can be constrained to be equal or variable across groups. Thus, 14 possible models with different geometric characteristics can be specified. Table 3 reports all such models with the corresponding distribution structure type, volume, shape, orientation, and associated model names. In Figure 2 the geometric characteristics are shown graphically.

Starting with version 5.0 of **mclust**, four additional models have been included: EVV, VEE, EVE, VVE. Models EVV and VEE are estimated using the methods described in Celeux and Govaert (1995), and the estimation of models EVE and VVE is carried out using the approach discussed by Browne and McNicholas (2014). In the models VEE, EVE and VVE it is assumed that the mixture components share the same orientation matrix. This assumption allows for a parsimonious characterisation of the clusters, while still retaining flexibility in defining volume and shape.

Model-based clustering

To illustrate the new modelling capabilities of **mclust** for model-based clustering consider the `wine` dataset contained in the **gclus** R package. This dataset provides 13 measurements obtained from a chemical analysis of 178 wines grown in the same region in Italy but derived from three different cultivars (Barolo, Grignolino, Barbera).

Model	$\boldsymbol{\Sigma}_k$	Distribution	Volume	Shape	Orientation
EII	$\lambda \boldsymbol{I}$	Spherical	Equal	Equal	—
VII	$\lambda_k \boldsymbol{I}$	Spherical	Variable	Equal	—
EEI	$\lambda \boldsymbol{A}$	Diagonal	Equal	Equal	Coordinate axes
VEI	$\lambda_k \boldsymbol{A}$	Diagonal	Variable	Equal	Coordinate axes
EVI	$\lambda \boldsymbol{A}_k$	Diagonal	Equal	Variable	Coordinate axes
VVI	$\lambda_k \boldsymbol{A}_k$	Diagonal	Variable	Variable	Coordinate axes
EEE	$\lambda \boldsymbol{D}\boldsymbol{A}\boldsymbol{D}^\top$	Ellipsoidal	Equal	Equal	Equal
EVE	$\lambda \boldsymbol{D}\boldsymbol{A}_k\boldsymbol{D}^\top$	Ellipsoidal	Equal	Variable	Equal
VEE	$\lambda_k \boldsymbol{D}\boldsymbol{A}\boldsymbol{D}^\top$	Ellipsoidal	Variable	Equal	Equal
VVE	$\lambda_k \boldsymbol{D}\boldsymbol{A}_k\boldsymbol{D}^\top$	Ellipsoidal	Variable	Variable	Equal
EEV	$\lambda \boldsymbol{D}_k\boldsymbol{A}\boldsymbol{D}_k^\top$	Ellipsoidal	Equal	Equal	Variable
VEV	$\lambda_k \boldsymbol{D}_k\boldsymbol{A}\boldsymbol{D}_k^\top$	Ellipsoidal	Variable	Equal	Variable
EVV	$\lambda \boldsymbol{D}_k\boldsymbol{A}_k\boldsymbol{D}_k^\top$	Ellipsoidal	Equal	Variable	Variable
VVV	$\lambda_k \boldsymbol{D}_k\boldsymbol{A}_k\boldsymbol{D}_k^\top$	Ellipsoidal	Variable	Variable	Variable

Table 3: Parameterisations of the within-group covariance matrix $\boldsymbol{\Sigma}_k$ for multidimensional data available in the **mclust** package, and the corresponding geometric characteristics.

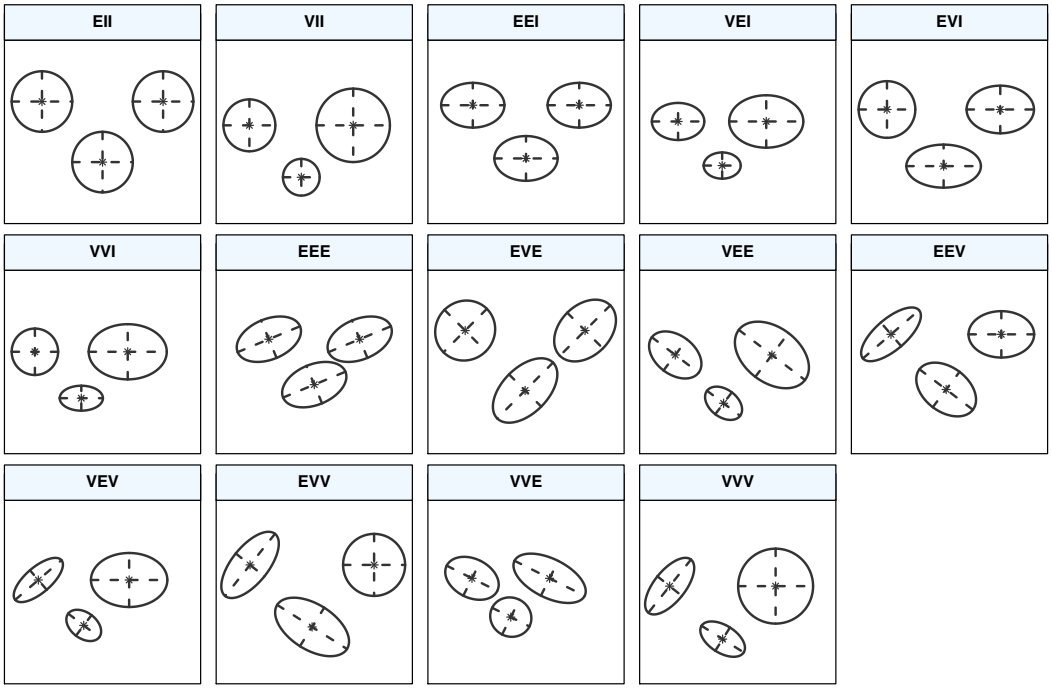


Figure 2: Ellipses of isodensity for each of the 14 Gaussian models obtained by eigen-decomposition in case of three groups in two dimensions.

```
> data(wine, package = "gclus")
> Class <- factor(wine$Class, levels = 1:3,
                  labels = c("Barolo", "Grignolino", "Barbera"))
> X <- data.matrix(wine[,-1])
> mod <- Mclust(X)
> summary(mod$BIC)
Best BIC values:
            EVE,3       VVE,3       VVE,6
BIC      -6873.257 -6896.83693 -6906.37460
BIC diff     0.000   -23.57947   -33.11714
> plot(mod, what = "BIC", ylim = range(mod$BIC[,-(1:2)], na.rm = TRUE),
       legendArgs = list(x = "bottomleft"))
```

In the above `Mclust()` function call, only the data matrix is provided, and the number of mixing

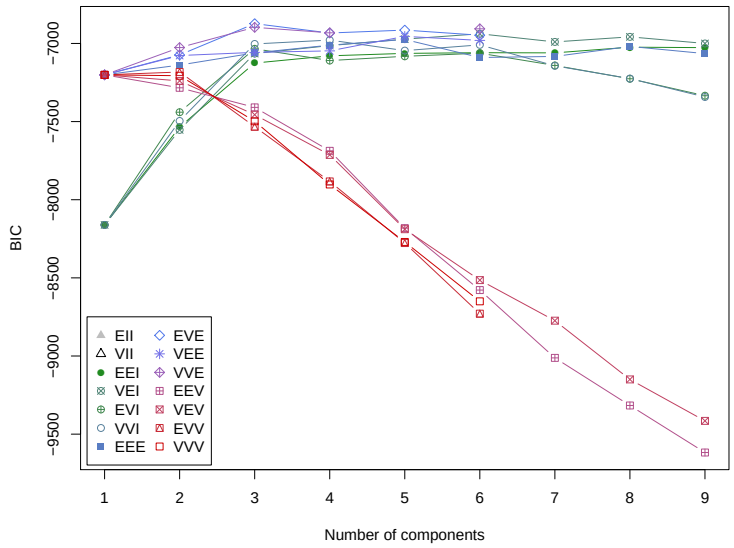


Figure 3: BIC plot for models fitted to the `wine` data.

components and the covariance parameterisation are selected using the Bayesian Information Criterion (BIC). A summary showing the top-three models and a plot of the BIC traces (see Figure 3) for all the models considered is then obtained. In the last plot we adjusted the range of the y-axis so to remove those models with lower BIC values. There is a clear indication of a three-component mixture with covariances having different shapes but the same volume and orientation (EVE). Note that all the top three models are among the models added to the latest major release of **mclust**.

A summary of the selected model is obtained as:

```
> summary(mod)
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------

Mclust EVE (ellipsoidal, equal volume and orientation) model with 3 components:

 log.likelihood   n  df       BIC       ICL
       -3032.45 178 156 -6873.257 -6873.549

Clustering table:
 1  2  3
63 51 64
```

The fitted model provides an accurate recovery of the true classes:

```
> table(Class, mod$classification)
Class         1  2  3
  Barolo     59  0  0
  Grignolino  4  3 64
  Barbera     0 48  0
> adjustedRandIndex(Class, mod$classification)
[1] 0.8803998
```

The latter index is the adjusted Rand index (ARI; Hubert and Arabie, 1985), which can be used for evaluating a clustering solution. The ARI is a measure of agreement between two partitions, one estimated by a statistical procedure independent of the labelling of the groups, and one being the true classification. It has zero expected value in the case of a random partition, and it is bounded above by 1, with higher values representing better partition accuracy.

To visualise the clustering structure and the geometric characteristics induced by an estimated Gaussian finite mixture model we may project the data onto a suitable dimension reduction subspace. The function `MclustDR()` implements the methodology introduced in Scrucca (2010). The estimated directions which span the reduced subspace are defined as a set of linear combinations of the original features, ordered by importance as quantified by the associated eigenvalues. By default, information on the dimension reduction subspace is provided by both the variation on cluster means and, depending

on the estimated mixture model, on the variation on cluster covariances. This methodology has been extended to supervised classification by Scrucca (2014). Furthermore, a tuning parameter has been included which enables the recovery of most of the separating directions, i.e. those that show maximal separation among groups. Other dimension reduction techniques for finding the directions of optimum separation have been discussed in detail by Hennig (2004) and implemented in the package **fpc** (Hennig, 2015).

Applying `MclustDR` to the wine data example, such directions are obtained as follows:

```
> drmod <- MclustDR(mod, lambda = 1)
> summary(drmod)
-----------------------------------------------------------------
Dimension reduction for model-based clustering and classification
-----------------------------------------------------------------

Mixture model type: Mclust (EVE, 3)

Clusters  n
       1 63
       2 51
       3 64

Estimated basis vectors:
                       Dir1       Dir2
Alcohol          0.11701058  0.2637302
Malic           -0.02814821  0.0489447
Ash             -0.18258917  0.5390056
Alcalinity      -0.02969793 -0.0309028
Magnesium        0.00575692  0.0122642
Phenols         -0.18497201 -0.0016806
Flavanoids       0.45479873 -0.2948947
Nonflavanoid     0.59278569 -0.5777586
Proanthocyanins  0.05347167  0.0508966
Intensity       -0.08328239  0.0332611
Hue              0.42950365 -0.4588969
OD280            0.40563746 -0.0369229
Proline          0.00075867  0.0010457

               Dir1    Dir2
Eigenvalues  1.5794   1.332
Cum. %      54.2499 100.000
```

By setting the optional tuning parameter `lambda = 1`, instead of the default value `0.5`, only the information on cluster means is used for estimating the directions. In this case, the dimension of the subspace is $d = \min(p, G-1)$, where p is the number of variables and G the number of mixture components or clusters. In the data example, there are $p = 13$ features and $G = 3$ clusters, so the dimension of the reduced subspace is $d = 2$. As a result, the projected data show the maximal separation among clusters, as shown in Figure 4a, which is obtained with

```
> plot(drmod, what = "contour")
```

On the same subspace we can also plot the uncertainty boundaries corresponding to the MAP classification:

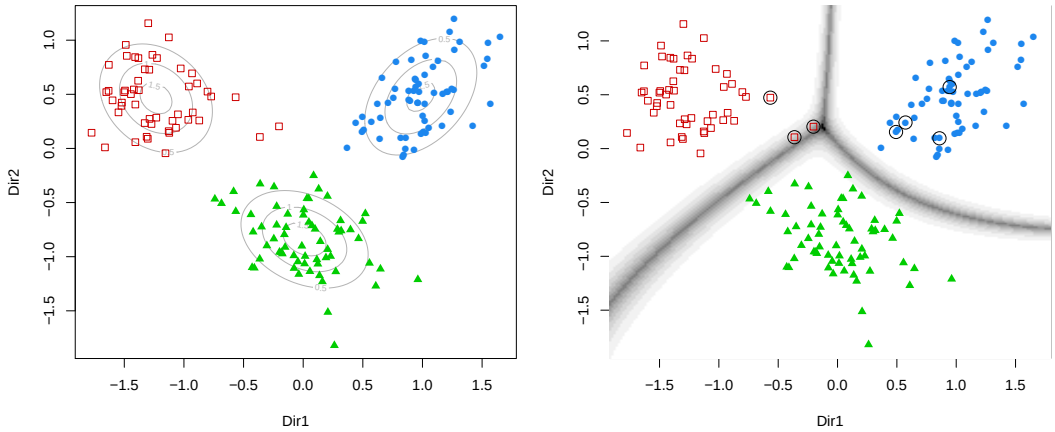
```
> plot(drmod, what = "boundaries", ngrid = 200)
```

and then add a circle around the misclassified observations

```
> miscl <- classError(Class, mod$classification)$misclassified
> points(drmod$dir[miscl,], pch = 1, cex = 2)
```

Model selection

A central question in finite mixture modelling is how many components should be included in the mixture. In GMMs we need also to decide which covariance parameterisation to adopt. Both questions can be addressed by information criteria, such as the BIC (Schwartz, 1978; Fraley and Raftery, 1998)



(a) (b)

Figure 4: Contour plot of estimated mixture densities (a) and uncertainty boundaries (b) on the projection subspace estimated with `MclustDR` for the `wine` dataset.

or the integrated complete-data likelihood criterion (ICL; Biernacki et al., 2000). The selection of the order of the mixture, i.e. the number of mixture components or clusters, can be also performed by formal hypothesis testing; for a recent review see McLachlan and Rathnayake (2014).

Information criteria are based on penalised forms of the log-likelihood. As the likelihood increases with the addition of more components, a penalty term for the number of estimated parameters is subtracted from the log-likelihood. The BIC is a popular choice in the context of GMMs, and takes the form

$$\text{BIC}_{\mathcal{M},G} = 2\ell_{\mathcal{M},G}(\boldsymbol{x}|\widehat{\boldsymbol{\Psi}}) - \nu \log(n),$$

where $\ell_{\mathcal{M},G}(\boldsymbol{x}|\widehat{\boldsymbol{\Psi}})$ is the log-likelihood at the MLE $\widehat{\boldsymbol{\Psi}}$ for model $\mathcal{M}$ with G components, n is the sample size, and ν is the number of estimated parameters. The pair $\{\mathcal{M}, G\}$ which maximises $\text{BIC}_{\mathcal{M},G}$ is selected. Given some necessary regularity conditions, BIC is derived as an approximation to the model evidence using the Laplace method. Although these conditions do not hold for mixture models in general (Aitkin and Rubin, 1985), some consistency results apply (Roeder and Wasserman, 1997; Keribin, 2000) and the criterion has been shown to perform well in applications (Fraley and Raftery, 1998).

In the **mclust** package, BIC is used by default for model selection. The function `mclustBIC()` allows the user to obtain a matrix of BIC values for all the available models and number of components up to 9 (by default).

For example, consider the `diabetes` dataset which contains measurements on 145 non-obese adult subjects. Recorded variables are `glucose`, the area under plasma glucose curve after a three hour oral glucose tolerance test (OGTT), `insulin`, the area under plasma insulin curve after a three hour OGTT, and `sspg`, the steady state plasma glucose level. The patients are classified clinically into three groups.

```
> data(diabetes)
> X <- diabetes[,2:4]
> Class <- diabetes$class
> table(Class)
Chemical   Normal    Overt
      36       76       33
```

The data can be shown graphically (see Figure 5) as follows:

```
> clp <- clPairs(X, Class, lower.panel = NULL)
> clPairsLegend(0.1, 0.3, class = clp$class, col = clp$col, pch = clp$pch)
```

The following function call can be used to compute the BIC for all the covariance structures and up to 9 components:

```
> BIC <- mclustBIC(X)
> BIC
Bayesian Information Criterion (BIC):
        EII       VII       EEI       VEI       EVI       VVI       EEE       EVE
1 -5863.923 -5863.923 -5530.129 -5530.129 -5530.129 -5530.129 -5136.446 -5136.446
2 -5449.518 -5327.719 -5169.399 -5019.350 -5015.884 -4988.322 -5010.994 -4875.633
```

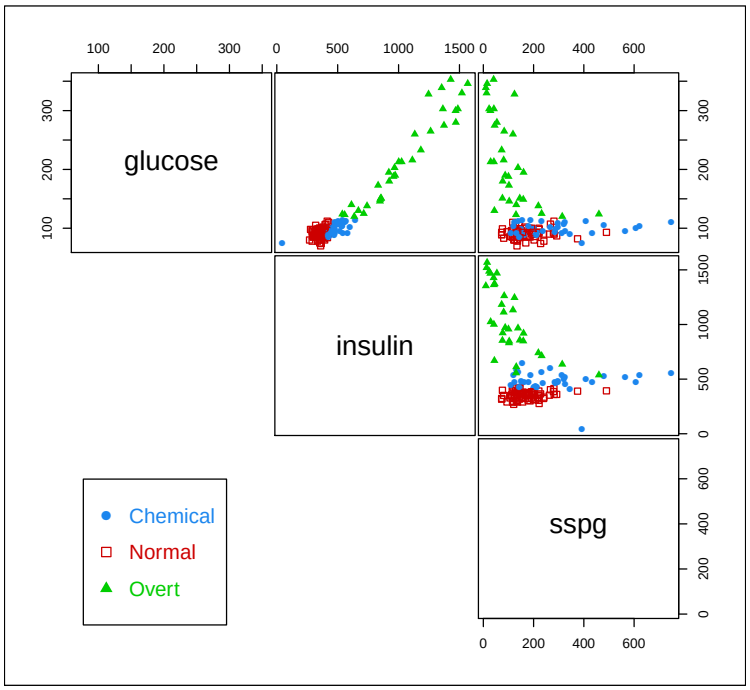


Figure 5: Pairwise scatterplots for the `diabetes` data with points marked according to classification.

```
3 -5412.588 -5206.399 -4998.446 -4899.759 -5000.661 -4827.818 -4976.853 -4858.851
4 -5236.008 -5208.512 -4937.627 -4835.856 -4865.767 -4813.002 -4865.864 -4793.261
5 -5181.608 -5202.555 -4915.486 -4841.773 -4838.587 -4833.589 -4882.812        NA
6 -5162.164 -5135.069 -4885.752        NA -4848.623 -4810.558 -4835.226        NA
7 -5128.736 -5129.460 -4857.097        NA -4849.023        NA -4805.518        NA
8 -5135.787 -5135.053 -4858.904        NA -4873.450        NA -4820.155        NA
9 -5150.374 -5112.616 -4878.786        NA -4865.166        NA -4840.039        NA
        VEE       VVE       EEV       VEV       EVV       VVV
1 -5136.446 -5136.446 -5136.446 -5136.446 -5136.446 -5136.446
2 -4920.301 -4877.086 -4918.500 -4834.727 -4823.779 -4825.027
3 -4851.667 -4775.537 -4917.567 -4809.225 -4817.884 -4760.091
4 -4840.034 -4794.892 -4887.406 -4823.882 -4828.796 -4802.420
5        NA        NA -4908.030 -4842.077        NA        NA
6        NA        NA -4844.584 -4826.457        NA        NA
7        NA        NA -4910.155 -4852.182        NA        NA
8        NA        NA -4858.974 -4870.633        NA        NA
9        NA        NA -4930.535 -4887.206        NA        NA

Top 3 models based on the BIC criterion:
    VVV,3     VVE,3     EVE,4
-4760.091 -4775.537 -4793.261
```

In the results reported above, the `NA` values mean that a particular model cannot be estimated. This happens in practice due to singularity in the covariance matrix estimate and can be avoided using the Bayesian regularisation proposed in Fraley and Raftery (2007a) and implemented in **mclust** as described in Fraley et al. (2012). Optional arguments allow finetuning, such as `G` for the number of components, and `modelNames` for specifying the model covariances parameterisations (see Table 3 and `help(mclustModelNames)` for a description of available model names). Another optional argument `x` can be used to provide the output from a previous call to `mclustBIC()`. This is useful if the model space needs to be enlarged by fitting more models, e.g. by increasing the number of mixture components, without the need to recompute the BIC values for those models already fitted. Another usage of such strategy that may be helpful to users is provided in `Mclust()`. For example, BIC values already available can be provided as follows

```
> Mclust(X, x = BIC)
```

Note that by specifying the argument `G` and `modelNames` the model space can be restricted to a subset, or enlarged to a superset. In the latter case the BIC is calculated only for the newly included models.

The use of BIC for model selection was available in **mclust** since earlier versions. However, BIC tends to select the number of mixture components needed to reasonably approximate the density, rather than the number of clusters as such. For this reason, other criteria have been proposed for model selection, like the *integrated complete-data likelihood* (ICL) criterion (Biernacki et al., 2000):

$$\mathrm{ICL}_{\mathcal{M},G} = \mathrm{BIC}_{\mathcal{M},G} + 2\sum_{i=1}^{n}\sum_{k=1}^{G} c_{ik}\log(z_{ik}),$$

where z_{ik} is the conditional probability that $\boldsymbol{x}_i$ arises from the kth mixture component, and $c_{ik} = 1$ if the ith unit is assigned to cluster k and 0 otherwise. ICL penalises the BIC through an *entropy* term which measures clusters overlap. Provided that clusters overlapping is not too strong, ICL has shown good performance in selecting the number of clusters, with preference for solutions with well-separated groups.

In **mclust** the ICL can be computed by means of the `mclustICL()` function:

```
> ICL <- mclustICL(X)
> ICL
Integrated Complete-data Likelihood (ICL) criterion:
        EII       VII       EEI       VEI       EVI       VVI       EEE       EVE
1 -5863.923 -5863.923 -5530.129 -5530.129 -5530.129 -5530.129 -5136.446 -5136.446
2 -5450.004 -5333.689 -5169.732 -5023.533 -5016.010 -4994.986 -5012.758 -4876.295
3 -5415.983 -5219.627 -4999.693 -4910.963 -5011.423 -4839.130 -4985.448 -4875.992
4 -5238.797 -5224.698 -4939.741 -4847.524 -4876.784 -4823.308 -4867.650 -4809.169
5 -5190.524 -5226.204 -4923.986 -4865.230 -4854.347 -4859.162 -4895.412        NA
6 -5171.561 -5158.411 -4901.823        NA -4865.106 -4820.076 -4846.827        NA
7 -5136.220 -5152.330 -4872.644        NA -4870.151        NA -4817.584        NA
8 -5146.628 -5156.135 -4871.975        NA -4897.172        NA -4834.074        NA
9 -5180.744 -5145.708 -4911.346        NA -4883.199        NA -4872.677        NA
        VEE       VVE       EEV       VEV       EVV       VVV
1 -5136.446 -5136.446 -5136.446 -5136.446 -5136.446 -5136.446
2 -4927.621 -4885.421 -4920.413 -4844.590 -4826.796 -4834.539
3 -4866.976 -4793.271 -4927.563 -4821.068 -4828.535 -4776.086
4 -4869.658 -4823.020 -4956.077 -4847.034 -4839.703 -4830.658
5        NA        NA -4948.787 -4869.279        NA        NA
6        NA        NA -4884.720 -4849.505        NA        NA
7        NA        NA -4947.190 -4878.445        NA        NA
8        NA        NA -4890.913 -4895.286        NA        NA
9        NA        NA -5007.250 -4919.228        NA        NA

Top 3 models based on the ICL criterion:
    VVV,3     VVE,3     EVE,4
-4776.086 -4793.271 -4809.169
```

As discussed above for `mclustBIC()`, the output from a previous call to `mclustICL()` can be provided as input with the argument `x` to avoid recomputing the ICL for models already fitted.

Both criteria can be shown graphically with (see Figure 6):

```
> plot(BIC)
> plot(ICL)
```

In this case BIC and ICL selected the same final model.

Other information criteria are available in the literature. For example, members of the Generalised Information Criteria (GIC) family (Konishi and Kitagawa, 1996) are not computed by the package, but they can be easily obtained using the information returned by the `Mclust()` function.

In addition to the information criteria just mentioned, the choice of the order of a mixture model for a specific component-covariances parameterisation can be carried out by likelihood ratio testing (LRT). Suppose we want to test the null hypothesis $H_0 : G = G_0$ against the alternative $H_1 : G = G_1$ for some $G_1 > G_0$; usually, $G_1 = G_0 + 1$ as it is a common procedure to keep adding components sequentially. Let $\widehat{\boldsymbol{\Psi}}_{G_j}$ be the MLE of $\boldsymbol{\Psi}$ calculated under $H_j : G = G_j$ (for $j = 0, 1$). The likelihood ratio test statistic (LRTS) can be written as

$$\mathrm{LRTS} = -2\log\{L(\widehat{\boldsymbol{\Psi}}_{G_0})/L(\widehat{\boldsymbol{\Psi}}_{G_1})\} = 2\{\ell(\widehat{\boldsymbol{\Psi}}_{G_1}) - \ell(\widehat{\boldsymbol{\Psi}}_{G_0})\},$$

where large values of LRTS provide evidence against the null hypothesis. However, standard regularity conditions do not hold for the null distribution of the LRTS to have its usual chi-squared distribution

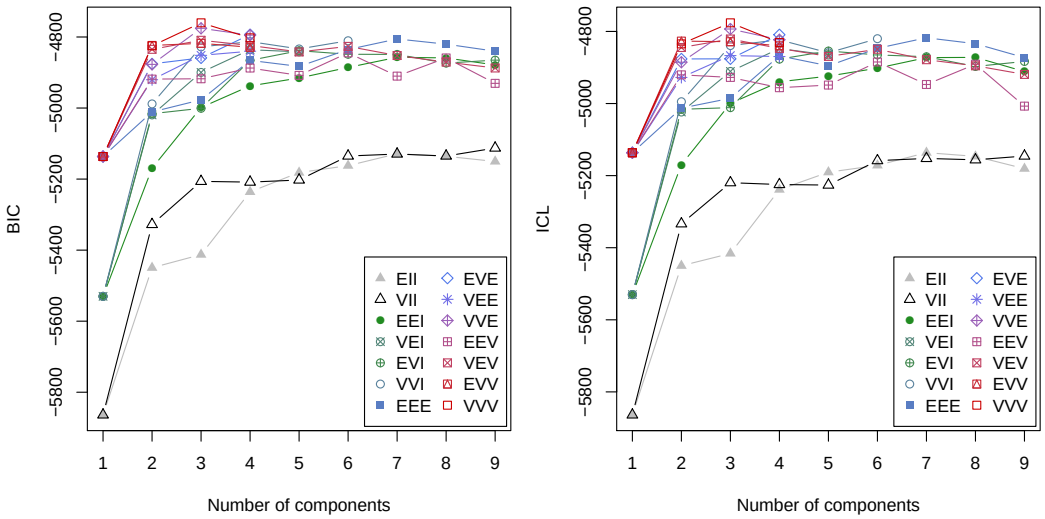


Figure 6: Plots of BIC and ICL model selection criteria for the `diabetes` data.

(McLachlan and Peel, 2000, Chap. 6). As consequence, LRT significance is often estimated by a resampling approach in order to produce a p-value. McLachlan (1987) proposed the using of the bootstrap to obtain the null distribution of the LRTS. The bootstrap procedure is the following:

1. a bootstrap sample $\boldsymbol{x}_b^*$ is generated by simulating from the fitted model under the null hypothesis with G_0 components, i.e. from the GMM distribution with the vector of unknown parameters replaced by MLEs obtained from the original data under H_0;
2. the test statistic LRTS_b^* is computed for the bootstrap sample $\boldsymbol{x}_b^*$ after fitting GMMs with G_0 and G_1 number of components;
3. steps 1. and 2. are replicated several times, say $B = 999$, to obtain the bootstrap null distribution of LRTS^*.

A bootstrap-based approximation to the p-value may then be computed as

$$p\text{-value} \approx \frac{1 + \sum_{i=1}^{B} \text{I}(\text{LRTS}_b^* \geq \text{LRTS}_{obs})}{B+1}$$

where LRTS_{obs} is the test statistic computed on the observed sample $\boldsymbol{x}$, and $\text{I}(\cdot)$ denotes the indicator function (which is equal to 1 if its argument is true and 0 otherwise).

The above bootstrap procedure is implemented in the `mclustBootstrapLRT()` function. We need to specify at least the input data and the model name we want to test:

```
> LRT <- mclustBootstrapLRT(X, modelName = "VVV")
> LRT
Bootstrap sequential LRT for the number of mixture components
--------------------------------------------------------------
Model        = VVV
Replications = 999
              LRTS bootstrap p-value
1 vs 2   361.186445             0.001
2 vs 3   114.703559             0.001
3 vs 4     7.437806             0.938
```

The number of bootstrap resamples can be set by the optional argument `nboot`; if not provided, `nboot = 999` is used. The sequential bootstrap procedure terminates when a test is not significant at the level specified by `level` (by default equal to `0.05`). There is also the option for a user to fix the maximum number of mixture components to test via the argument `maxG`. In the example above the bootstrap p-values clearly indicate the presence of three clusters. Note that models fitted on the original data are estimated via the EM algorithm initialised by the default model-based hierarchical agglomerative clustering. Then, during the bootstrap procedure, models under the null and the alternative hypotheses are fitted on bootstrap samples using again the EM algorithm. However, in this case the algorithm starts with the E step initialised with the estimated parameters obtained at the convergence of the EM algorithm on the original data.

The bootstrap distributions of the LRTS can be shown graphically (see Figure 7) using the associated plot method:

```
> plot(LRT, G = 1)
> plot(LRT, G = 2)
> plot(LRT, G = 3)
```

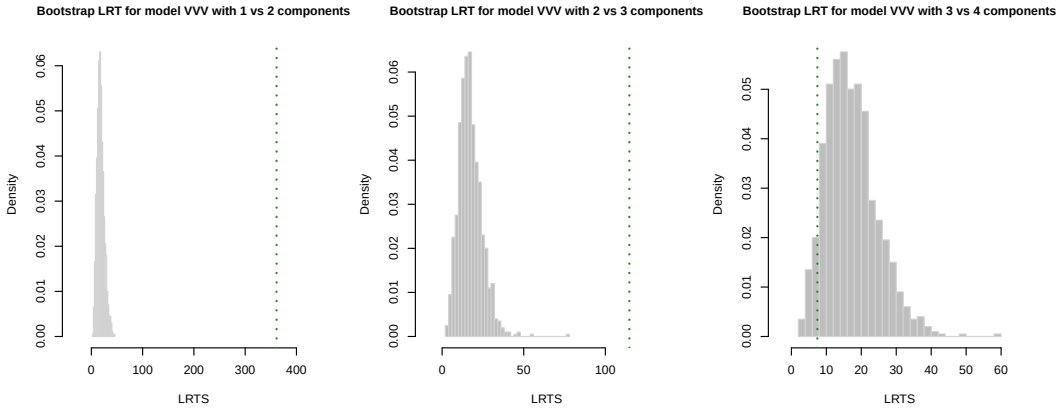


Figure 7: Histograms of LRTS bootstrap distributions for testing the number of mixture components in the `diabetes` data. The dotted vertical lines refer to the sample values of LRTS.

Bootstrap inference

There are two main approaches to likelihood-based inference in mixture models, namely information-based and resampling methods (McLachlan and Peel, 2000). In information-based methods, the covariance matrix of the MLE $\widehat{\boldsymbol{\Psi}}$ is approximated by the inverse of the observed information matrix $I^{-1}(\widehat{\boldsymbol{\Psi}})$, i.e.

$$\text{Cov}(\widehat{\boldsymbol{\Psi}}) \approx (I^{-1}(\widehat{\boldsymbol{\Psi}})).$$

However, "the sample size n has to be very large before the asymptotic theory applies to mixture models" (McLachlan and Peel, 2000, p. 42). Indeed, Basford et al. (1997) found that standard errors obtained using the expected or the observed information matrix are unstable, unless the sample size is very large. For these reasons, they advocate the use of a resampling approach based on the bootstrap. For a recent review and comparison of different resampling approaches to inference in finite mixture models see O'Hagan et al. (2015).

The *bootstrap* (Efron, 1979) is a general, widely applicable, powerful technique for obtaining an approximation to the sampling distribution of a statistic of interest. The bootstrap distribution is approximated by drawing a large number of samples (*bootstrap samples*) from the empirical distribution, i.e. by resampling with replacement from the observed data (*nonparametric bootstrap*), or from a parametric distribution with unknown parameters substituted by the corresponding estimates (*parametric bootstrap*).

Let $\widehat{\boldsymbol{\Psi}}$ be the estimate of a set of GMM parameters $\boldsymbol{\Psi}$ for a given model $\mathcal{M}$, i.e. covariance parameterisation, and number of mixture components G. A bootstrap estimate of the corresponding standard errors can be obtained using the following procedure:

- Obtain the bootstrap distribution for the parameters of interest by:
  1. drawing a sample of size n with replacement from the empirical distribution $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ to form the bootstrap sample $(\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_n^*)$;
  2. fitting a GMM $(\mathcal{M}, G)$ to get the bootstrap estimates $\widehat{\boldsymbol{\Psi}}^*$;
  3. replicating steps 1–2 a large number of times, say B, to obtain $\widehat{\boldsymbol{\Psi}}_1^*, \widehat{\boldsymbol{\Psi}}_2^*, \ldots, \widehat{\boldsymbol{\Psi}}_B^*$ estimates from B resamples.
- The bootstrap covariance matrix is then approximated by

$$\text{Cov}_{\text{boot}}(\widehat{\boldsymbol{\Psi}}) \approx \frac{1}{B-1} \sum_{b=1}^{B} (\widehat{\boldsymbol{\Psi}}_b^* - \overline{\widehat{\boldsymbol{\Psi}}}^*)(\widehat{\boldsymbol{\Psi}}_b^* - \overline{\widehat{\boldsymbol{\Psi}}}^*)^\top$$

where $\overline{\widehat{\boldsymbol{\Psi}}}^* = \frac{1}{B}\sum_{b=1}^{B}\widehat{\boldsymbol{\Psi}}_b^*$.

- The bootstrap standard errors for the parameter estimates $\widehat{\boldsymbol{\Psi}}$ are computed as the square root of the diagonal elements of the bootstrap covariance matrix, i.e.

$$\mathrm{se}_{\mathrm{boot}}(\widehat{\boldsymbol{\Psi}}) = \sqrt{\mathrm{diag}(\mathrm{Cov}_{\mathrm{boot}}(\widehat{\boldsymbol{\Psi}}))}.$$

Consider the `hemophilia` dataset (Habbema et al., 1974) available in the package **rrcov**, which contains two measured variables on 75 women belonging to two groups: 30 of them are non-carriers (normal group) and 45 are known hemophilia A carriers (obligatory carriers).

```
> data(hemophilia, package = "rrcov")
> X <- hemophilia[,1:2]
> Class <- as.factor(hemophilia$gr)
> plot(X, pch = ifelse(Class == "normal", 1, 16))
> legend("bottomright", legend = levels(Class), pch = c(16,1), inset = 0.03)
```

The last command plots the observed data marked by the known classification (see Figure 8a).

In analogy with the analysis of Basford et al. (1997, example II, Sec. 5), we fitted a two-components GMM with unconstrained covariance matrices:

```
> mod <- Mclust(X, G = 2, modelName = "VVV")
> summary(mod, parameters = TRUE)
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------

Mclust VVV (ellipsoidal, varying volume, shape, and orientation) model with 2 components:

 log.likelihood  n df      BIC      ICL
       77.02852 75 11 106.5647 92.85533

Clustering table:
 1  2
39 36

Mixing probabilities:
        1         2
0.5108084 0.4891916

Means:
                   [,1]        [,2]
AHFactivity -0.11627884 -0.36656353
AHFantigen  -0.02457577 -0.04534792

Variances:
[,,1]
            AHFactivity AHFantigen
AHFactivity  0.01137602 0.00659927
AHFantigen   0.00659927 0.01239353
[,,2]
            AHFactivity AHFantigen
AHFactivity  0.01585986 0.01505449
AHFantigen   0.01505449 0.03236079
```

Note that in the `summary()` function call we used the optional argument `parameters = TRUE` to retrieve the estimated parameters.

The clustering structure identified is shown in Figure 8b and can be obtained as follows:

```
> plot(mod, what = "classification", main = FALSE)
```

Bootstrap inference for GMMs is available through the function `MclustBootstrap()`, which requires the user to input an object returned by a call to `Mclust()`. Optionally, the user can also provide the number of bootstrap resamples `nboot` and the `type` of bootstrap to perform. By default, `nboot = 999` and `type = "bs"` for the nonparametric bootstrap. Thus, a simple call for computing the bootstrap distribution of the GMM parameters is the following:

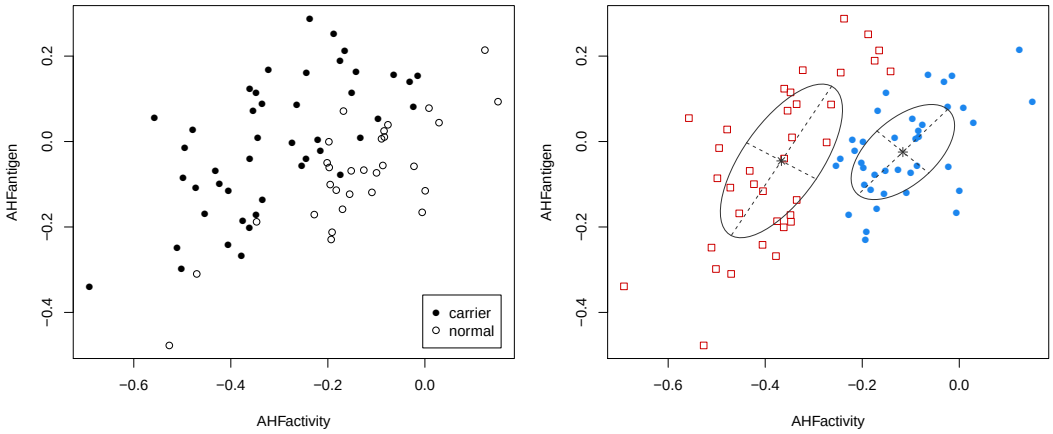


Figure 8: True class membership (a) and estimated classification using GMM (b) for the `hemophilia` dataset.

```
> boot <- MclustBootstrap(mod, nboot = 999, type = "bs")
```

Note that for the sake of clarity we have included the arguments `nboot` and `type`, but they can be omitted since they are set at their defaults.

The function `MclustBootstrap()` returns an object which can be plotted or summarised. For instance, to graph the bootstrap distribution for the mixing proportions and for the component means we may use the code:

```
> par(mfrow = c(1,2))
> plot(boot, what = "pro")
> par(mfrow = c(2,2))
> plot(boot, what = "mean")
> par(mfrow = c(1,1))
```

The resulting plots are shown, respectively, in Figures 9 and 10.

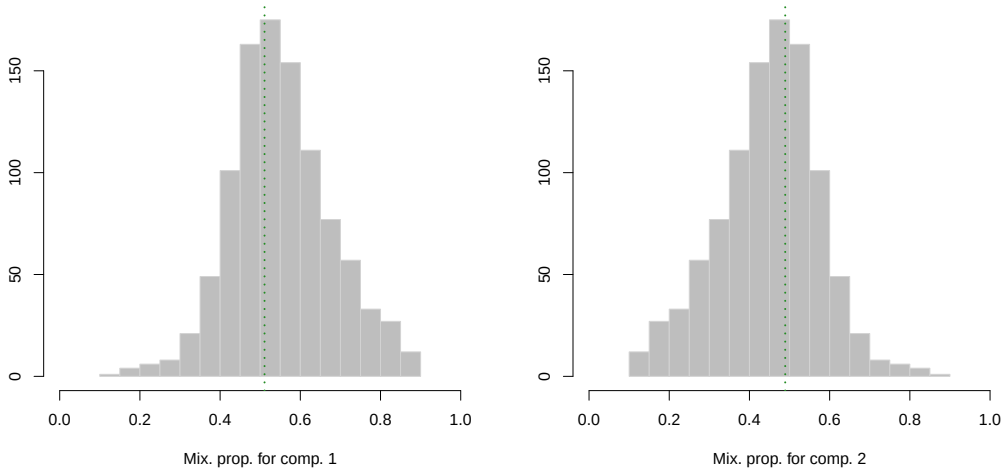


Figure 9: Bootstrap distribution for the mixture proportions. The vertical dotted lines refer to the MLEs for the GMM fitted to the `hemophilia` data.

A numerical summary of the bootstrap procedure is available through the `summary` method, which by default returns the standard errors of GMM parameters:

```
> summary(boot, what = "se")
---------------------------------------------------------
Resampling standard errors
---------------------------------------------------------
Model                       = VVV
```

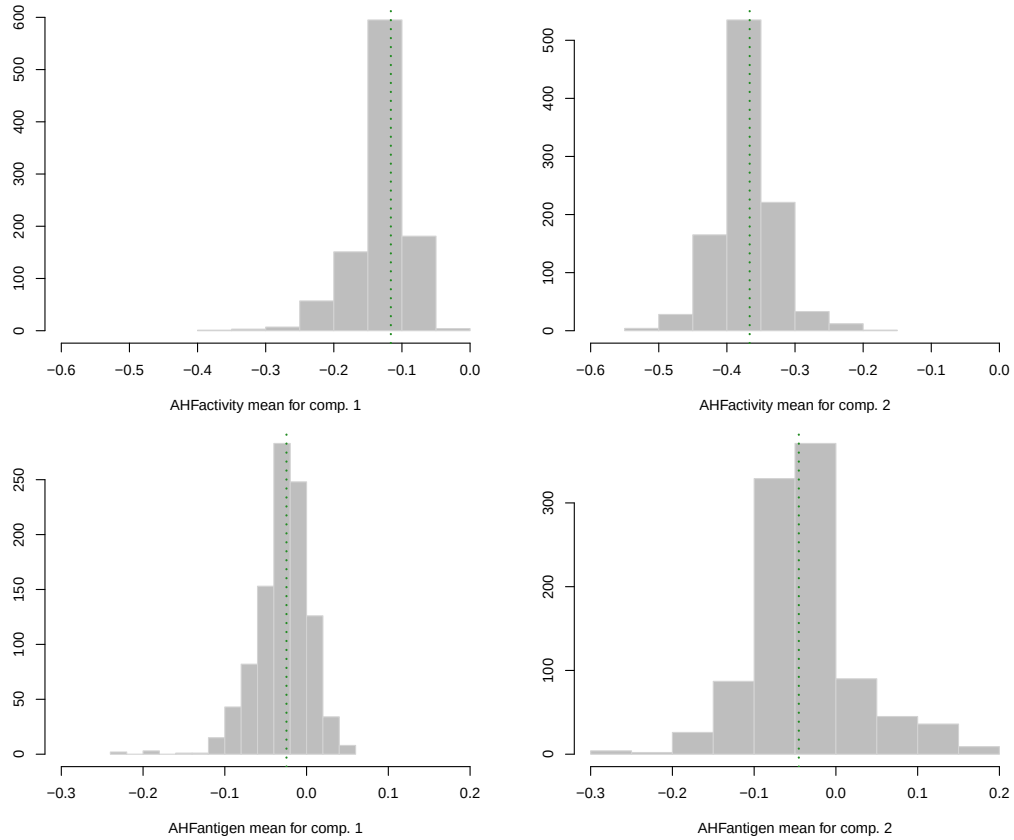


Figure 10: Bootstrap distribution for the mixture component means. The vertical dotted lines refer to the MLEs for the GMM fitted to the `hemophilia` data.

```
Num. of mixture components = 2
Replications               = 999
Type                       = nonparametric bootstrap

Mixing probabilities:
        1         2
0.1249357 0.1249357

Means:
                    1          2
AHFactivity 0.04028375 0.04137370
AHFantigen  0.03262182 0.06456482

Variances:
[,,1]
            AHFactivity  AHFantigen
AHFactivity 0.007018580 0.004690481
AHFantigen  0.004690481 0.003155312
[,,2]
            AHFactivity  AHFantigen
AHFactivity 0.005757398 0.005897374
AHFantigen  0.005897374 0.009654623
```

The `summary` method can also return bootstrap percentile confidence intervals. For the generic GMM parameter ψ of $\mathbf{\Psi}$, the percentile method yields the intervals $[\psi^*_{\alpha/2}, \psi^*_{1-\alpha/2}]$, where ψ^*_q is the qth quantile (or the $100q$th percentile) of the bootstrap distribution $(\widehat{\psi}^*_1, \ldots, \widehat{\psi}^*_B)$. These can be obtained by specifying in the `summary` call the argument `what = "ci"` and, optionally, the confidence level of the intervals (by default, `conf.level = 0.95`). For instance:

```
> summary(boot, what = "ci")
-------------------------------------------------------------
```

```
Resampling confidence intervals
---------------------------------------------------------
Model                      = VVV
Num. of mixture components = 2
Replications               = 999
Type                       = nonparametric bootstrap
Confidence level           = 0.95

Mixing probabilities:
              1         2
2.5%  0.3193742 0.1785054
97.5% 0.8214946 0.6806258

Means:
[,,1]
      AHFactivity  AHFantigen
2.5%  -0.22915526 -0.09784996
97.5% -0.07315876  0.02481681
[,,2]
      AHFactivity AHFantigen
2.5%   -0.4573113 -0.1571624
97.5%  -0.2747451  0.1318332

Variances:
[,,1]
      AHFactivity  AHFantigen
2.5%  0.004743597 0.007012672
97.5% 0.032144767 0.019245540
[,,2]
      AHFactivity  AHFantigen
2.5%  0.003981163 0.006049076
97.5% 0.027297495 0.045854646
```

The function `MclustBootstrap()` has also the provision for using the weighted likelihood bootstrap (Newton and Raftery, 1994). This is a generalisation of the nonparametric bootstrap which assigns random (positive) weights to sample observations; it can be viewed as a generalized Bayesian bootstrap. The weights are obtained from a uniform Dirichlet distribution, i.e. by sampling from n independent standard exponential distributions and then rescaling by their average. Then, the function `me.weighted()` in **mclust** allows one to apply a weighted EM algorithm. This approach may yield benefits when one or more components have small mixture proportions. In that case, a nonparametric bootstrap sample may have no representatives of them, but the weighted likelihood bootstrap will always have representatives of all groups.

In our data example the weighted likelihood bootstrap can be easily obtained by specifying `type = "wlbs"` in the `MclustBootstrap()` function call:

```
> wlboot <- MclustBootstrap(mod, nboot = 999, type = "wlbs")
> summary(wlboot, what = "se")
---------------------------------------------------------
Resampling standard errors
---------------------------------------------------------
Model                      = VVV
Num. of mixture components = 2
Replications               = 999
Type                       = weighted likelihood bootstrap

Mixing probabilities:
        1         2
0.1323612 0.1323612

Means:
                    1          2
AHFactivity 0.03977347 0.04192182
AHFantigen  0.02989056 0.06897928

Variances:
```

```
[,,1]
            AHFactivity  AHFantigen
AHFactivity 0.007074450 0.004686432
AHFantigen  0.004686432 0.003254011
[,,2]
            AHFactivity  AHFantigen
AHFactivity 0.005511614 0.005746981
AHFantigen  0.005746981 0.009883791
```

In this case the differences between the nonparametric and the weighted likelihood bootstrap are negligible. We can summarise the inference for the components means obtained under the two approaches with the following graphs of bootstrap percentile confidence intervals:

```
> boot.ci <- summary(boot, what = "ci")
> wlboot.ci <- summary(wlboot, what = "ci")
> par(mfrow = c(1,2), mar = c(4,4,1,1))
> for(j in 1:mod$G)
    { plot(1:mod$G, mod$parameters$mean[j,], col = 1:mod$G, pch = 15,
           ylab = colnames(X)[j], xlab = "Mixture component",
           ylim = range(boot.ci$mean,wlboot.ci$mean),
           xlim = c(.5,mod$G+.5), xaxt = "n")
     points(1:mod$G+0.2, mod$parameters$mean[j,], col = 1:mod$G, pch = 15)
     axis(side = 1, at = 1:mod$G)
     with(boot.ci, errorBars(1:G, mean[1,j,], mean[2,j,], col = 1:G))
     with(wlboot.ci, errorBars(1:G+0.2, mean[1,j,], mean[2,j,], col = 1:G, lty = 2))
  }
> par(mfrow = c(1,1))
```

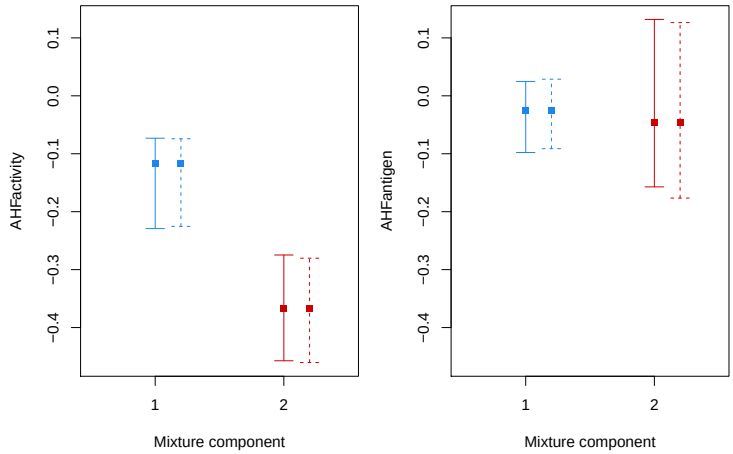


Figure 11: Bootstrap percentile intervals for the means of the GMM fitted to the `hemophilia` dataset. Solid lines refer to nonparametric bootstrap, dashed lines to the weighted likelihood bootstrap.

Initialisation of the EM algorithm

The EM algorithm is an easy to implement and numerically stable algorithm which has reliable global convergence under fairly general conditions. However, the likelihood surface in mixture models tends to have multiple modes and thus initialisation of EM is crucial because it usually produces sensible results when started from reasonable starting values (Wu, 1983).

In **mclust** the EM algorithm is initialised using the partitions obtained from model-based hierarchical agglomerative clustering (MBHAC). In this approach, hierarchical clusters are obtained by recursively merging the two clusters that provide the smallest decrease in the classification likelihood for Gaussian mixture model (Banfield and Raftery, 1993). Efficient numerical algorithms have been discussed by Fraley (1998). Using MBHAC is particularly convenient because the underlying probabilistic model is shared by both the initialisation step and the model fitting step. Furthermore, MBHAC is also computationally advantageous because a single run provides the basis for initialising the EM algorithm for any number of mixture components and component-covariances parameterisa-

tions. Although there is no guarantee that the EM initialized by MBHAC will converge to the global optimum, it often provides reasonable starting points.

A problem with the MBHAC approach may arise in the presence of coarse data, resulting from the discrete nature of the data or from continuous data that are rounded when measured. In this case, ties must be broken by choosing the pair of entities that will be merged. This is often done at random, but regardless of which method is adopted for breaking ties, this choice can have important consequences because it changes the clustering of the remaining observations. Moreover, the final EM solution may depend on the ordering of the variables.

Consider the Flea beetles data available in package **tourr**. This dataset provides six physical measurements for a sample of 72 flea beetles from three species:

```
> data(flea, package = "tourr")
> X <- data.matrix(flea[,1:6])
> Class <- factor(flea$species, labels = c("Concinna","Heikertingeri","Heptapotamica"))
> table(Class)
Class
     Concinna Heikertingeri Heptapotamica
           21            31            22
> col <- mclust.options("classPlotColors")[1:3]
> clp <- clPairs(X, Class, lower.panel = NULL, gap = 0,
                 symbols = c(16,15,17), colors = adjustcolor(col, alpha.f = 0.5))
> clPairsLegend(x = 0.1, y = 0.3, class = clp$class, col = col, pch = clp$pch,
              title = "Flea beatle species")
```

As can be seen from Figure 12, the observed values are rounded (to the nearest integer presumably) and there is a strong overplotting of points.

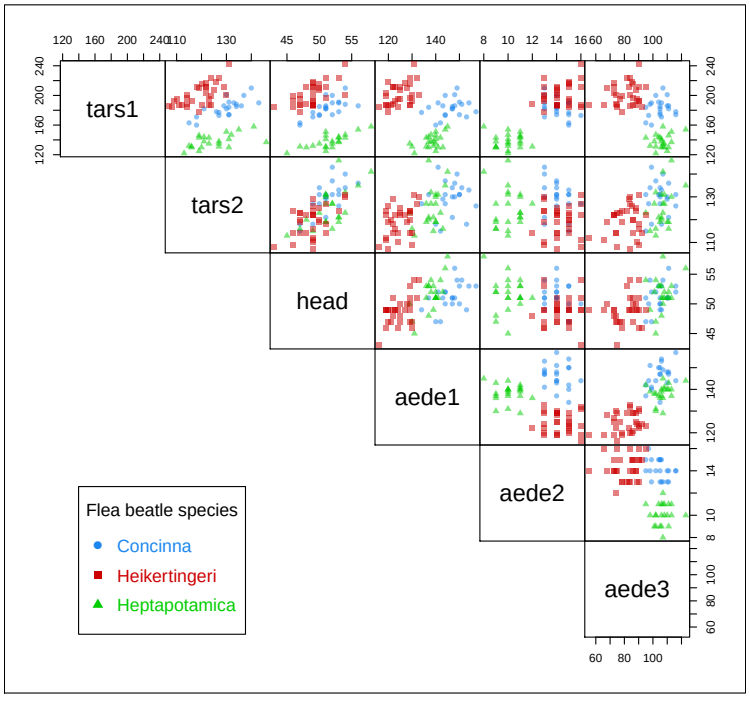


Figure 12: Scatterplot matrix for the Flea beetles data with points marked according to the true classes.

```
> mod1 <- Mclust(X)
> summary(mod1)
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------

Mclust EEE (ellipsoidal, equal volume, shape and orientation) model with 5 components:

 log.likelihood  n df       BIC       ICL
      -1292.308 74 55 -2821.339 -2825.769
```

```
Clustering table:
 1  2  3  4  5
21  2 20 20 11
> adjustedRandIndex(Class, mod1$classification)
[1] 0.7675713

> mod2 <- Mclust(X[,6:1])
> summary(mod2)
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------

Mclust EEE (ellipsoidal, equal volume, shape and orientation) model with 5 components:

 log.likelihood  n df       BIC       ICL
      -1287.027 74 55 -2810.777 -2812.702

Clustering table:
 1  2  3  4  5
22 21 22  7  2
> adjustedRandIndex(Class, mod2$classification)
[1] 0.8131206
```

By reversing the order of the variables in the fit of `mod2`, the initial partitions differ due to ties in the data, so the EM algorithm converges to different solutions of the same EEE model with 5 components. The second solution has a higher BIC and better accuracy.

In situations like this we may want to assess the stability of results by randomly starting the EM algorithm. The function `randomPairs()` may be called to obtain a random hierarchical structure suitable to be used as initial clustering partition:

```
> mod3 <- Mclust(X, initialization = list(hcPairs = randomPairs(X, seed = 123)))
> summary(mod3)
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------

Mclust EEE (ellipsoidal, equal volume, shape and orientation) model with 4 components:

 log.likelihood  n df       BIC       ICL
      -1298.211 74 48 -2803.017 -2807.713

Clustering table:
 1  2  3  4
16 15 22 21
> adjustedRandIndex(Class, mod3$classification)
[1] 0.7867056
```

Using a random start we obtain a EEE model with 4 components, which has a higher BIC but a lower ARI. However, a better initialisation may be found using the approach discussed in Scrucca and Raftery (2015). The main idea is to project the data through a suitable transformation which enhances separation among clusters before applying the MBHAC at the initialisation step. Once a reasonable hierarchical partition is obtained, the EM algorithm is run using the data on the original scale. For instance, a GMM started using the scaled SVD transformation is obtained with the following code:

```
> mod4 <- Mclust(X, initialization = list(hcPairs = hc(X, use = "SVD")))
> summary(mod4)
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------

Mclust EEE (ellipsoidal, equal volume, shape and orientation) model with 3 components:

 log.likelihood  n df       BIC       ICL
      -1304.552 74 41 -2785.572 -2785.574

Clustering table:
```

```
 1  2  3
21 31 22
> adjustedRandIndex(Class, mod4$classification)
[1] 1
```

In this case we achieve both the highest BIC and a perfect classification of the fleas into the actual species.

We conclude by noting that in the case of large datasets, i.e. having a large number of observations or cases, a subsample of the data can be used in the MBHAC phase before applying the EM algorithm to the full data set. This is easily done by providing an optional argument to `Mclust()` or `mclustBIC()` (as well as many other functions) as a vector, say `s`, of logical values or numerical indices specifying the subset of data to be used in the initial hierarchical clustering phase:

```
> Mclust(X, initialization = list(subset = s))
```

Density estimation

Density estimation plays an important role in applied statistical data analysis and theoretical research. Finite mixture models provide a flexible semi-parametric model-based approach to density estimation, which makes it possible to accurately approximate any given probability distribution. **mclust** provides a simple interface to Gaussian mixture models for univariate and multivariate density estimation.

Izenman and Sommer (1988) considered the fitting of a Gaussian mixture to the distribution of the thickness of stamps in the 1872 Hidalgo stamp issue of Mexico [2]. A density estimate based on GMM can be obtained using the function `densityMclust()`:

```
> data(Hidalgo1872, package = "MMST")
> Thickness <- Hidalgo1872$thickness
> Year <- rep(c("1872", "1873-74"), c(289, 196))
> dens <- densityMclust(Thickness)
> summary(dens$BIC)
Best BIC values:
             V,3         V,5        V,4
BIC      2983.791 2974.939223 2972.19349
BIC diff    0.000   -8.852019  -11.59775
> summary(dens, parameters = TRUE)
-------------------------------------------------------
Density estimation via Gaussian finite mixture modeling
-------------------------------------------------------

Mclust V (univariate, unequal variance) model with 3 components:

 log.likelihood   n df      BIC      ICL
       1516.632 485  8 2983.791 2890.914

Clustering table:
  1   2   3
128 171 186

Mixing probabilities:
        1         2         3
0.2661410 0.3011217 0.4327374

Means:
         1          2          3
0.07215458 0.07935341 0.09919740

Variances:
             1              2              3
0.000004814927 0.000003097694 0.000188461484
```

[2] The Hidalgo stamp data is available at the home page for the book by Izenman (2008) at http://astro.temple.edu/~alan/MMST/datasets.html, or through the package **MMST**. The latter has been archived on CRAN, so it must be installed using the following code:

```
> install.packages("http://cran.r-project.org/src/contrib/Archive/MMST/MMST_0.6-1.1.tar.gz", repos
= NULL, type = "source")
```

The model selected is a three-component mixture with different variances. A graph of the density estimated is shown in Figure13a and is obtained with the code:

```
> br <- seq(min(Thickness), max(Thickness), length = 21)
> plot(dens, what = "density", data = Thickness, breaks = br)
```

Here a histogram of the observed data is also drawn by providing the optional argument `data` and with breakpoints between histogram cells specified in the argument `breaks`. From the graph, three modes appear at the means of the mixture components: one with larger stamp thickness, and two corresponding to thinner stamps.

Additional information can also be used. In particular, thickness measurements can be grouped according to the year of consignment; the first 289 stamps refer to the 1872 issue, and the remaining 196 stamps to the years 1873–1874. We may draw a (suitable scaled) histogram for each year-of-consignment and then add the estimated components densities as follows:

```
> h1 <- hist(Thickness[Year == "1872"], breaks = br, plot = FALSE)
> h1$density <- h1$density*prop.table(table(Year))[1]
> h2 <- hist(Thickness[Year == "1873-74"], breaks = br, plot = FALSE)
> h2$density <- h2$density*prop.table(table(Year))[2]
> x <- seq(min(Thickness)-diff(range(Thickness))/10,
           max(Thickness)+diff(range(Thickness))/10, length = 200)
> cdens <- predict(dens, x, what = "cdens")
> cdens <- t(apply(cdens, 1, function(d) d*dens$parameters$pro))
> col <- adjustcolor(mclust.options("classPlotColors")[1:2], alpha = 0.3)
> plot(h1, xlab = "Thickness", freq = FALSE, main = "", border = FALSE, col = col[1],
       xlim = range(x), ylim = range(h1$density, h2$density, cdens))
> plot(h2, add = TRUE, freq = FALSE, border = FALSE, col = col[2])
> matplot(x, cdens, type = "l", lwd = 1, add = TRUE, lty = 1:3, col = 1)
> box()
```

The result is shown in Figure 13b. Stamps from 1872 show a two-regime distribution, with one corresponding to the component with the largest thickness, and one whose distribution essentially overlaps with the bimodal distribution of stamps for the years 1873–1874.

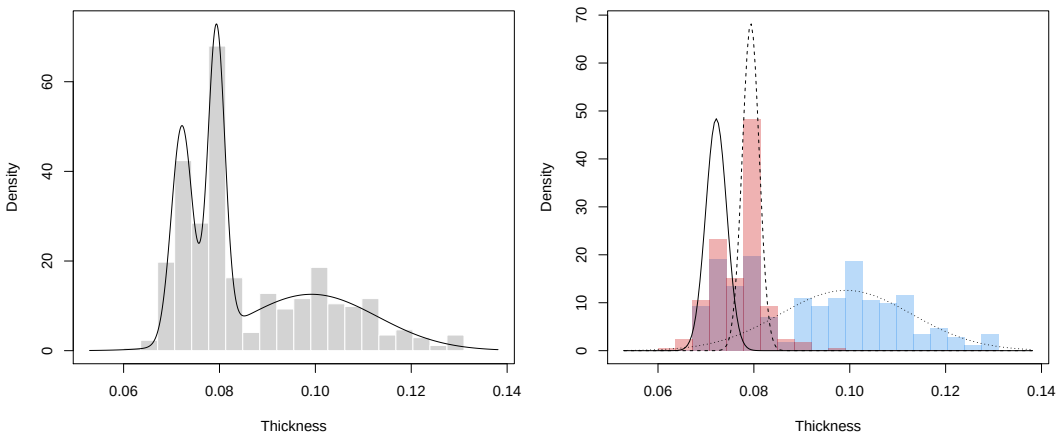


Figure 13: (a) Histogram with mixture-based density estimate curve, and (b) histograms by group-year with estimated mixture-component densities, for the `Hidalgo1872` stamps dataset.

As an example of bivariate density estimation, consider the well-known 'Old Faithful' data set which provides the waiting time between eruptions (`waiting`) and the duration of the eruptions (`eruptions`) for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA. The dataset can be read and data plotted as follows:

```
> data(faithful)
> plot(faithful, cex = 0.5)
```

A bivariate density estimate for the Faithful data is obtained with the commands:

```
> dens <- densityMclust(faithful)
> summary(dens)
```

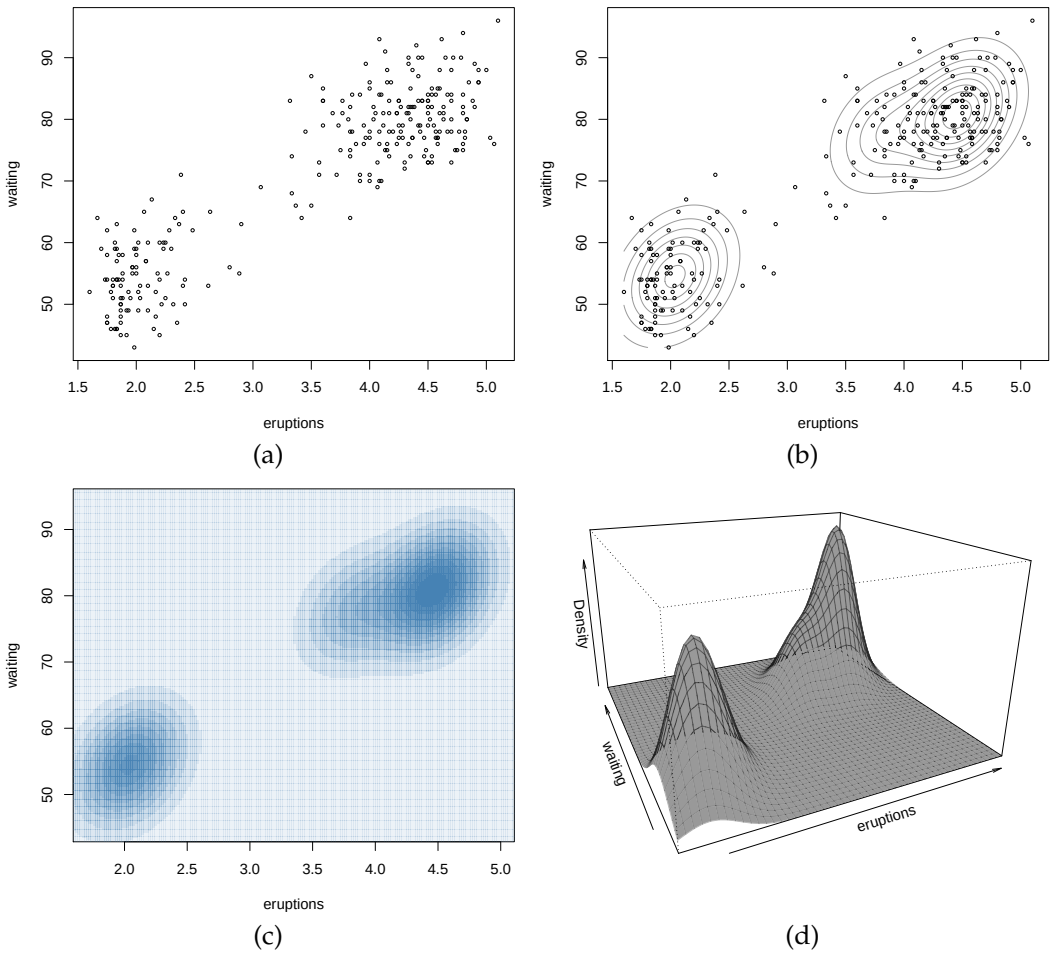


Figure 14: Plot of the Old Faithful data (a), mixture-based density estimate contours (b), image plot of density estimate (c) and perspective plot of the bivariate density estimate (d).

```
-------------------------------------------------------
Density estimation via Gaussian finite mixture modeling
-------------------------------------------------------

Mclust EEE (ellipsoidal, equal volume, shape and orientation) model with 3 components:

 log.likelihood   n df       BIC       ICL
      -1126.361 272 11 -2314.386 -2360.865

Clustering table:
  1   2   3
130  97  45
```

Model selection based on the BIC selects a three-component mixture with common covariance matrix (EEE). One component is used to model the group of observations having both low duration and low waiting times, whereas two components are needed to approximate the skewed distribution of the observations with larger duration and waiting times.

Figure 14b-d shows some of the available graphs in **mclust** for a bivariate density estimated by GMM. These can be obtained with the commands:

```
> plot(dens, what = "density", data = faithful, grid = 200, points.cex = 0.5,
       drawlabels = FALSE)
> plot(dens, what = "density", type = "image", col = "steelblue", grid = 200)
> plot(dens, what = "density", type = "persp", theta = -25, phi = 20,
       border = adjustcolor(grey(0.1), alpha.f = 0.3))
```

Note that the same procedure using the function `mclustDensity()` can also be used to obtain density estimates for higher dimensional datasets.

Supervised classification

In supervised classification or discriminant analysis the aim is to build a classifier (or a decision rule) which is able to assign an observation with an unknown class membership to one of K known classes. For building a supervised classifier, a training dataset $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ is used for which both the features $\boldsymbol{x}_i$ and true classes $y_i \in \{C_1, \ldots, C_K\}$ are known.

Mixture-based discriminant analysis models assume that the density for each class follows a Gaussian mixture distribution

$$f_k(\boldsymbol{x}) = \sum_{g=1}^{G_k} \pi_{gk} \phi(\boldsymbol{x}; \boldsymbol{\mu}_{gk}, \boldsymbol{\Sigma}_{gk}),$$

where π_{gk} are the mixing probabilities for class k ($\pi_{gk} > 0, \sum_{g=1}^{G_k} \pi_{gk} = 1$), $\boldsymbol{\mu}_{gk}$ the means for component g within class k, and $\boldsymbol{\Sigma}_{gk}$ the covariance matrix of component g within class k. Hastie and Tibshirani (1996) proposed Mixture Discriminant Analysis (MDA) where it is assumed that the covariance matrix is the same for all the classes but is otherwise unconstrained, i.e. $\boldsymbol{\Sigma}_{gk} = \boldsymbol{\Sigma}$ for all g and k. The number of mixture components is assumed known for each class.

Bensmail and Celeux (1996) proposed the Eigenvalue Decomposition Discriminant Analysis (EDDA) which assumes that the density for each class can be described by a single Gaussian component (i.e. $G_k = 1$ for all k) with the component covariance structure factorised as

$$\boldsymbol{\Sigma}_k = \lambda_k \boldsymbol{D}_k \boldsymbol{A}_k \boldsymbol{D}_k^\top.$$

Several models can be obtained from the above decomposition. If $\boldsymbol{\Sigma}_k = \lambda \boldsymbol{D}\boldsymbol{A}\boldsymbol{D}^\top$ (model EEE), then EDDA is equivalent to linear discriminant analysis (LDA). If $\boldsymbol{\Sigma}_k = \lambda_k \boldsymbol{D}_k \boldsymbol{A}_k \boldsymbol{D}_k^\top$ (model VVV) then EDDA is equivalent to quadratic discriminant analysis (QDA).

Consider the UCI Wisconsin breast cancer diagnostic data available at http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic). This dataset provides data for 569 patients on 30 features of the cell nuclei obtained from a digitized image of a fine needle aspirate (FNA) of a breast mass (Mangasarian et al., 1995). For each patient the cancer was diagnosed as malignant or benign. Following Fraley and Raftery (2002) we considered only three attributes: extreme area, extreme smoothness, and mean texture. The dataset can be downloaded from the UCI repository using the following commands:

```
> data <- read.csv("http://archive.ics.uci.edu/ml/machine-learning-databases/breast-
cancer-wisconsin/wdbc.data", header = FALSE)
> X <- data[,c(4, 26, 27)]
> colnames(X) <- c("texture.mean", "area.extreme", "smoothness.extreme")
> Class <- data[,2]
```

Then, we may randomly assign approximately 2/3 of the observations to the training set, and the remaining ones to the test set:

```
> set.seed(123)
> train <- sample(1:nrow(X), size = round(nrow(X)*2/3), replace = FALSE)
> X.train <- X[train,]
> Class.train <- Class[train]
> table(Class.train)
Class.train
  B   M
238 141
> X.test <- X[-train,]
> Class.test <- Class[-train]
> table(Class.test)
Class.test
  B   M
119  7 1
```

The function `MclustDA()` provides fitting capabilities for the EDDA model, but we must specify the optional argument `modelType = "EDDA"`. The function call is thus the following:

```
> mod1 <- MclustDA(X.train, Class.train, modelType = "EDDA")
> summary(mod1, newdata = X.test, newclass = Class.test)
------------------------------------------------
Gaussian finite mixture model for classification
------------------------------------------------
```

```
EDDA model summary:

 log.likelihood   n df       BIC
      -2989.967 379 12 -6051.185

Classes   n Model G
      B 238   VVI 1
      M 141   VVI 1

Training classification summary:

     Predicted
Class   B   M
    B 237   1
    M  19 122

Training error = 0.05277045

Test classification summary:

     Predicted
Class   B   M
    B 116   3
    M   5  66

Test error = 0.04210526
```

The EDDA mixture model selected by BIC is the VVI model, so each group is described by a single Gaussian component with varying volume and shape, but same orientation aligned with the coordinate axes. Note that in the `summary()` function call we also provided the features and the known classes for the test set, so both the training error and the test error are reported. A cross-validation error can also be computed using the `cvMclustDA()` function, which by default use `nfold = 10` for a 10-fold cross-validation:

```
> cv <- cvMclustDA(mod1)
> unlist(cv[c("error", "se")])
      error          se
0.052770449 0.007930516
```

EDDA imposes a single mixture component for each group. However, in certain circumstances more complexity may improve performance. A more general approach, called *MclustDA*, has been proposed by Fraley and Raftery (2002), where a finite mixture of Gaussian distributions is used within each class, with number of components and covariance matrix structures (expressed following the usual decomposition) being different between classes. This is the default model fitted by `MclustDA`:

```
> mod2 <- MclustDA(X.train, Class.train)
> summary(mod2, newdata = X.test, newclass = Class.test)
------------------------------------------------
Gaussian finite mixture model for classification
------------------------------------------------

MclustDA model summary:

 log.likelihood   n df       BIC
      -2937.586 379 29 -6047.361

Classes   n Model G
      B 238   EEV 2
      M 141   VVI 2

Training classification summary:

     Predicted
Class   B   M
    B 236   2
```

```
    M   7 134

Training error = 0.0237467

Test classification summary:

     Predicted
Class   B   M
    B 114   5
    M   2  69

Test error = 0.03684211
```

A two-component mixture distribution is fitted to both the benign and malignant observations, but with different covariance structures within each class. Both the training error and the test error are slightly smaller than for EDDA, a fact also confirmed by the 10-fold cross-validation procedure:

```
> cv <- cvMclustDA(mod2)
> unlist(cv[c("error", "se")])
      error          se
0.021108179 0.007648168
```

A plot method which produces a variety of graphs is associated with objects returned by `MclustDA`. For instance, pairwise scatterplots between the features, showing both the known classes and the estimated mixture components, are drawn as follows (see Figure 15a–c):

```
> plot(mod2, what = "scatterplot", dimens = c(1,2))
> plot(mod2, what = "scatterplot", dimens = c(2,3))
> plot(mod2, what = "scatterplot", dimens = c(3,1))
```

Another interesting graph can be obtained by projecting the data on a dimension reduced subspace (Scrucca, 2014) with the commands:

```
> drmod2 <- MclustDR(mod2)
> summary(drmod2)
-----------------------------------------------------------------
Dimension reduction for model-based clustering and classification
-----------------------------------------------------------------

Mixture model type: MclustDA

Classes   n Model G
      B 238   EEV 2
      M 141   VVI 2

Estimated basis vectors:
                         Dir1         Dir2          Dir3
texture.mean       -0.00935540 -0.044384467 -0.0006607120
area.extreme        0.00049997  0.000071676 -0.0000088494
smoothness.extreme  0.99995611 -0.999014521  0.9999997817

               Dir1     Dir2       Dir3
Eigenvalues  0.67718  0.28159   0.013928
Cum. %      69.61869 98.56810 100.000000
> plot(drmod2, what = "boundaries", ngrid = 200)
```

The graph produced by the last command is shown in Figure 15d. The two groups are largely separated along the first direction, with the group of malignant cases showing a higher variability.

Finally, note that the MDA model is equivalent to MclustDA with $\boldsymbol{\Sigma}_k = \lambda \boldsymbol{D}\boldsymbol{A}\boldsymbol{D}^\top$ (model EEE) and fixed $G_k \geq 1$ for each $k = 1, \ldots, K$. For instance, a MDA with two mixture components for each class can be fitted as:

```
> mod3 <- MclustDA(X.train, Class.train, G = 2, modelNames = "EEE")
> summary(mod3, newdata = X.test, newclass = Class.test)
-----------------------------------------------
Gaussian finite mixture model for classification
-----------------------------------------------
```

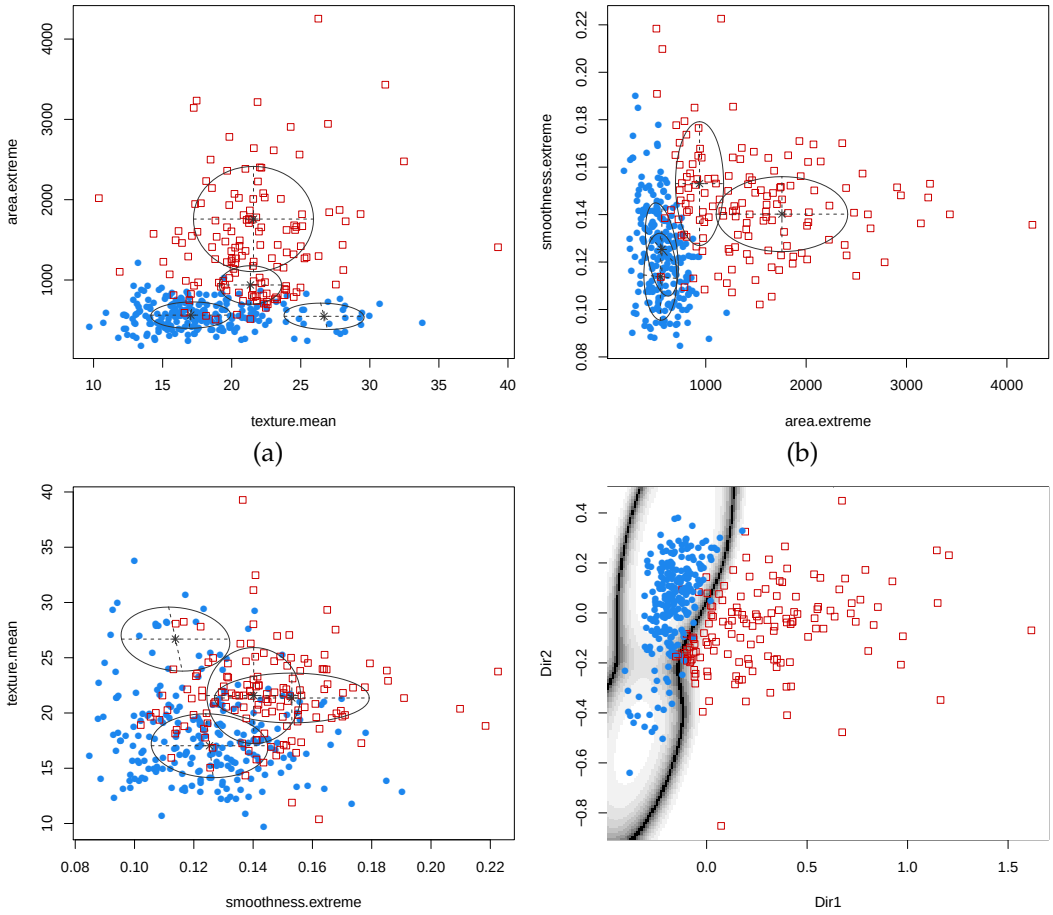


Figure 15: Pairwise scatterplots between variables for the Wisconsin breast cancer data (panels a–c). Points are marked by cancer diagnosis (benign = ●, malignant = □), whereas ellipses correspond to covariances of mixture components estimated with `MclustDA`. Plot of data projected along the first two estimated directions obtained with `MclustDR`, and uncertainty classification boundaries (d).

```
MclustDA model summary:

 log.likelihood   n df       BIC
      -2968.077 379 26 -6090.531

Classes   n Model G
      B 238   EEE 2
      M 141   EEE 2

Training classification summary:

     Predicted
Class   B   M
    B 235   3
    M  12 129

Training error = 0.03957784

Test classification summary:

     Predicted
Class   B   M
    B 113   6
```

```
    M   2  69

Test error = 0.04210526
```

Summary

mclust is one of the most popular R packages for Gaussian mixture modelling. Since its early developments (Banfield and Raftery, 1993; Fraley and Raftery, 1998, 1999), **mclust** has seen major updates through the years, which expanded its capabilities and features, increasing its popularity and widening its area of utilisation.

Here we have presented the most salient new features introduced in version ≥ 5, namely new covariance parameterisations, subspace data visualisation, different model selection criteria, bootstrap-based inference and EM algorithm initialisation. We showed their application on a collection of different datasets, pointing out their utility in different contexts.

Acknowledgments

Michael Fop and T. Brendan Murphy were supported by the Science Foundation Ireland funded Insight Research Centre (SFI/12/RC/2289). Adrian E. Raftery and Luca Scrucca were supported by NIH grants R01 HD054511, R01 HD070936 and U54 HL127624.

Bibliography

J. S. Ahlquist and C. Breunig. Model-based clustering and typologies in the social sciences. *Political Analysis*, 20(1):92–112, 2012. [p289]

M. Aitkin and D. B. Rubin. Estimation and hypothesis testing in finite mixture models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 47(1):67–75, 1985. [p295]

J. Banfield and A. E. Raftery. Model-based Gaussian and non-Gaussian clustering. *Biometrics*, 49: 803–821, 1993. [p291, 304, 314]

K. E. Basford, D. R. Greenway, G. J. McLachlan, and D. Peel. Standard errors of fitted component means of normal mixtures. *Computational Statistics*, 12(1):1–18, 1997. [p299, 300]

T. Benaglia, D. Chauveau, D. R. Hunter, and D. Young. mixtools: An R package for analyzing finite mixture models. *Journal of Statistical Software*, 32(6):1–29, 2009. URL http://www.jstatsoft.org/v32/i06/. [p289]

H. Bensmail and G. Celeux. Regularized Gaussian discriminant analysis through eigenvalue decomposition. *Journal of the American Statistical Association*, 91:1743–1748, 1996. [p310]

P. Biecek, E. Szczurek, M. Vingron, and J. Tiuryn. The R package bgmm: Mixture modeling with uncertain knowledge. *Journal of Statistical Software*, 47(3):1–32, 2012. URL http://www.jstatsoft.org/v47/i03/. [p289]

C. Biernacki, G. Celeux, and G. Govaert. Assessing a mixture model for clustering with the integrated completed likelihood. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(7):719–725, 2000. [p295, 297]

S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the Seventh International Conference on World Wide Web*, pages 107–117, 1998. [p290]

R. P. Browne and P. D. McNicholas. Estimating common principal components in high dimensions. *Advances in Data Analysis and Classification*, 8(2):217–226, 2014. [p291]

R. P. Browne, A. ElSherbiny, and P. D. McNicholas. *mixture: Mixture Models for Clustering and Classification*, 2015. URL https://CRAN.R-project.org/package=mixture. R package version 1.4. [p289]

J. G. Campbell, C. Fraley, D. Stanford, F. Murtagh, and A. E. Raftery. Model-based methods for textile fault detection. *International Journal of Imaging Systems and Technology*, 10(4):339–346, 1999. [p289]

G. Celeux and G. Govaert. Gaussian parsimonious clustering models. *Pattern Recognition*, 28:781–793, 1995. [p291]

W.-C. Chen and R. Maitra. *EMCluster: EM Algorithm for Model-Based Clustering of Finite Mixture Gaussian Distribution*, 2015. URL https://CRAN.R-project.org/package=EMCluster. R package version 0.2-5. [p289]

G. Csardi. *cranlogs: Download Logs from the RStudio CRAN Mirror*, 2015. URL https://github.com/metacran/cranlogs. R package version 2.0.0. [p289]

G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL http://igraph.org. [p290]

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 39(1):1–38, 1977. [p291]

B. Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7:1–26, 1979. [p299]

K. J. Ellefsen, D. B. Smith, and J. D. Horton. A modified procedure for mixture-model clustering of regional geochemical data. *Applied Geochemistry*, 51:315–326, 2014. [p289]

A. Flynt and M. I. G. Daepp. Diet-related chronic disease in the northeastern United States: A model-based clustering approach. *International Journal of Health Geographics*, 14(1):1–14, 2015. [p289]

C. Fraley. Algorithms for model-based Gaussian hierarchical clustering. *SIAM Journal on Scientific Computing*, 20(1):270–281, 1998. [p304]

C. Fraley and A. E. Raftery. How many clusters? Which clustering method? Answers via model-based cluster analysis. *The Computer Journal*, 41:578–588, 1998. [p294, 295, 314]

C. Fraley and A. E. Raftery. MCLUST: Software for model-based cluster analysis. *Journal of Classification*, 16(2):297–306, 1999. [p290, 314]

C. Fraley and A. E. Raftery. Model-based clustering, discriminant analysis, and density estimation. *Journal of the American Statistical Association*, 97(458):611–631, 2002. [p310, 311]

C. Fraley and A. E. Raftery. Enhanced model-based clustering, density estimation, and discriminant analysis software: Mclust. *Journal of Classification*, 20(2):263–286, 2003. [p290]

C. Fraley and A. E. Raftery. Some applications of model-based clustering in chemistry. *R News*, 6(3): 17–23, 2006a. URL http://CRAN.R-project.org/doc/Rnews/Rnews_2006-3.pdf. [p289]

C. Fraley and A. E. Raftery. Model-based microarray image analysis. *R News*, 6(5):60–63, 2006b. URL http://CRAN.R-project.org/doc/Rnews/Rnews_2006-5.pdf. [p289]

C. Fraley and A. E. Raftery. Bayesian regularization for normal mixture estimation and model-based clustering. *Journal of Classification*, 24(2):155–181, 2007a. [p296]

C. Fraley and A. E. Raftery. Model-based methods of classification: Using the mclust software in chemometrics. *Journal of Statistical Software*, 18(6):1–13, 2007b. URL http://www.jstatsoft.org/v018/i06/. [p289]

C. Fraley, A. E. Raftery, T. B. Murphy, and L. Scrucca. MCLUST version 4 for R: Normal mixture modeling for model-based clustering, classification, and density estimation. Technical Report 597, Department of Statistics, University of Washington, 2012. [p290, 296]

C. Fraley, A. E. Raftery, and L. Scrucca. *mclust: Gaussian Mixture Modelling for Model-Based Clustering, Classification, and Density Estimation*, 2016. URL https://CRAN.R-project.org/package=mclust. R package version 5.2. [p289]

B. Grün and F. Leisch. Fitting finite mixtures of generalized linear regressions in R. *Computational Statistics & Data Analysis*, 51(11):5247–5252, 2007. [p289]

B. Grün and F. Leisch. FlexMix version 2: Finite mixtures with concomitant variables and varying and constant parameters. *Journal of Statistical Software*, 28(4):1–35, 2008. URL http://www.jstatsoft.org/v28/i04/. [p289]

J. D. F. Habbema, J. Hermans, and K. van den Broek. A stepwise discriminant analysis program using density estimation. In *Proceedings in Computational Statistics*, pages 101–110, Vienna: Physica-Verlag, 1974. COMPSTAT. [p300]

T. Hastie and R. Tibshirani. Discriminant analysis by Gaussian mixtures. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 58(1):155–176, 1996. [p310]

C. Hennig. Asymmetric linear dimension reduction for classification. *Journal of Computational and Graphical Statistics*, 13(4):930–945, 2004. [p294]

C. Hennig. *fpc: Flexible Procedures for Clustering*, 2015. URL https://CRAN.R-project.org/package=fpc. R package version 2.1-10. [p294]

L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2:193–218, 1985. [p293]

C. Hurley. *gclus: Clustering Graphics*, 2012. URL https://CRAN.R-project.org/package=gclus. R package version 1.3.1. [p291]

A. J. Izenman. *Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning*. Springer-Verlag, New York, 2008. [p307]

A. J. Izenman and C. J. Sommer. Philatelic mixtures and multimodal densities. *Journal of the American Statistical Association*, 83(404):941–953, 1988. [p307]

J. Jang and D. B. Hitchcock. Model-based cluster analysis of democracies. *Journal of Data Science*, 10(2): 297–319, 2012. [p289]

K. Kazor and A. S. Hering. Assessing the performance of model-based clustering methods in multivariate time series with application to identifying regional wind regimes. *Journal of Agricultural, Biological and Environmental Statistics*, 20:192–217, 2015. [p289]

C. Keribin. Consistent estimation of the order of mixture models. *Sankhyā: The Indian Journal of Statistics, Series A (1961–2002)*, 62(1):49–66, 2000. [p295]

K. H. Kim, S. T. Yun, S. S. Park, Y. Joo, and T. S. Kim. Model-based clustering of hydrochemical data to demarcate natural versus human impacts on bedrock groundwater quality in rural areas, South Korea. *Journal of Hydrology*, 519:626–636, 2014. [p289]

L. W. Konigsberg, B. F. B. Algee-Hewitt, and D. W. Steadman. Estimation and evidence in forensic anthropology: Sex and race. *American Journal of Physical Anthropology*, 139:77–90, 2009. [p289]

S. Konishi and G. Kitagawa. Generalised information criteria in model selection. *Biometrika*, 83(4): 875–890, 1996. [p297]

M. Kozak and C. H. Scaman. Unsupervised classification methods in food sciences: Discussion and outlook. *Journal of the Science of Food and Agriculture*, 88(7):1115–1127, 2008. [p289]

R. Lebret, S. Iovleff, F. Langrognet, C. Biernacki, G. Celeux, and G. Govaert. Rmixmod: The R package of the model-based unsupervised, supervised, and semi-supervised classification Mixmod library. *Journal of Statistical Software*, 67(6):1–29, 2015. URL http://www.jstatsoft.org/v067/i06/. [p289]

F. Leisch. FlexMix: A general framework for finite mixture models and latent class regression in R. *Journal of Statistical Software*, 11(8):1–18, 2004. URL http://www.jstatsoft.org/v11/i08/. [p289]

Q. Li, C. Fraley, R. E. Bumgarner, and A. E. Raftery. Donuts, scratches and blanks: Robust model-based segmentation of microarray images. *Bioinformatics*, 21:2875–2882, 2005. [p289]

O. L. Mangasarian, W. N. Street, and W. H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, 43(4):570–577, 1995. [p310]

G. McLachlan and D. Peel. *Finite Mixture Models*. Wiley, New York, 2000. [p291, 298, 299]

G. J. McLachlan. On bootstrapping the likelihood ratio test statistic for the number of components in a normal mixture. *Applied Statistics*, 36:318–324, 1987. [p298]

G. J. McLachlan and S. Rathnayake. On the number of components in a Gaussian mixture model. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(5):341–355, 2014. [p295]

M. A. Newton and A. E. Raftery. Approximate Bayesian inference with the weighted likelihood bootstrap (with discussion). *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 56:3–48, 1994. [p303]

A. O'Hagan, T. Brendan Murphy, and I. C. Gormley. On estimation of parameter uncertainty in model-based clustering. *ArXiv e-prints*, oct 2015. URL http://arxiv.org/abs/1510.00551. [p299]

K. Roeder and L. Wasserman. Practical bayesian density estimation using mixtures of normals. *Journal of the American Statistical Association*, 92(439):894–902, 1997. [p295]

G. Schwartz. Estimating the dimension of a model. *The Annals of Statistics*, 6:31–38, 1978. [p294]

L. Scrucca. Dimension reduction for model-based clustering. *Statistics and Computing*, 20(4):471–484, 2010. [p293]

L. Scrucca. Graphical tools for model-based mixture discriminant analysis. *Advances in Data Analysis and Classification*, 8(2):147–165, 2014. [p294, 312]

L. Scrucca and A. E. Raftery. Improved initialisation of model-based clustering using Gaussian hierarchical partitions. *Advances in Data Analysis and Classification*, 4(9):447–460, 2015. [p306]

C. Suveg, M. L. Jacob, M. Whitehead, A. Jones, and J. N. Kingery. A model-based cluster analysis of social experiences in clinically anxious youth: links to emotional functioning. *Anxiety, Stress, & Coping*, 27(5):494–508, 2014. [p289]

M. Templ, P. Filzmoser, and C. Reimann. Cluster analysis applied to regional geochemical data – problems and possibilities. *Applied Geochemistry*, 23:2198–2213, 2008. [p289]

V. Todorov and P. Filzmoser. An object-oriented framework for robust multivariate analysis. *Journal of Statistical Software*, 32(3):1–47, 2009. URL http://www.jstatsoft.org/v32/i03/. [p291]

B. Verbist, L. Clement, J. Reumers, K. Thys, A. Vapirev, W. Talloen, Y. Wetzels, J. Meys, J. Aerssens, L. Bijnens, and O. Thas. ViVaMBC: Estimating viral sequence variation in complex populations from illumina deep-sequencing data using model-based clustering. *BMC Bioinformatics*, 16(1):1–11, 2015. [p289]

H. Wickham, D. Cook, H. Hofmann, and A. Buja. tourr: An R package for exploring multivariate data with projections. *Journal of Statistical Software*, 40(2):1–18, 2011. URL http://www.jstatsoft.org/v40/i02/. [p291]

C. J. Wu. On the convergence properties of the EM algorithm. *The Annals of Statistics*, 11(1):95–103, 1983. [p304]

K. Y. Yeung, C. Fraley, A. Murua, A. E. Raftery, Raftery, and W. L. Ruzzo. Model-based clustering and data transformations for gene expression data. *Bioinformatics*, 17(10):977–987, 2001. [p289]

Luca Scrucca
Università degli Studi di Perugia
Via A. Pascoli 20, 06123 Perugia
Italy
luca.scrucca@unipg.it

Michael Fop
University College Dublin
Belfield, Dublin 4
Ireland
michael.fop@ucdconnect.ie

T. Brendan Murphy
University College Dublin
Belfield, Dublin 4
Ireland
brendan.murphy@ucd.ie

Adrian E. Raftery
University of Washington
Box 354320
Seattle, WA 98195-4320
raftery@u.washington.edu

clustering.sc.dp: Optimal Clustering with Sequential Constraint by Using Dynamic Programming

by Tibor Szkaliczki

Abstract The general clustering algorithms do not guarantee optimality because of the hardness of the problem. Polynomial-time methods can find the clustering corresponding to the exact optimum only in special cases. For example, the dynamic programming algorithm can solve the one-dimensional clustering problem, i.e., when the items to be clustered can be characterised by only one scalar number. Optimal one-dimensional clustering is provided by package **Ckmeans.1d.dp** in R. The paper shows a possible generalisation of the method implemented in this package to multidimensional data: the dynamic programming method can be applied to find the optimum clustering of vectors when only subsequent items may form a cluster. Sequential data are common in various fields including telecommunication, bioinformatics, marketing, transportation etc. The proposed algorithm can determine the optima for a range of cluster numbers in order to support the case when the number of clusters is not known in advance.

Introduction

Clustering plays a key role in various areas including data mining, character recognition, information retrieval, machine learning applied in diverse fields such as marketing, medicine, engineering, computer science, etc. A clustering algorithm forms groups of similar items in a data set which is a crucial step in analysing complex data. Clustering can be formulated as an optimisation problem assigning items to clusters while minimising the distances among the cluster members. The normally used clustering algorithms do usually not find the optimal solution because the clustering problem is NP-complete in the general case. This paper introduces a package implementing an optimisation method for clustering in a special case when a sequential constraint should be met, i.e., when the items to be clustered are sorted and only subsequent items may form a cluster. This constraint is common when clustering data streams, e.g., audio and video streams, trajectories, motion tracks, click-streams etc. The good news is that the exact optimum can be found in polynomial time in this case.

The algorithm recommended for clustering sequential data is based on the dynamic programing approach developed by Wang and Song (2011). They gave a polynomial-time algorithm for one-dimensional clustering, i.e., when the items can be characterised by only one scalar number. Similarly to the heuristic k-means algorithm, it divides data into k groups and it minimises the within-cluster sum of squared distances (WCSS or *withinss* for short). The algorithm guarantees a solution minimising the optimisation goal. The source code of the algorithm is available in the R package **Ckmeans.1d.dp** (Song and Wang, 2011). The generalisation of the algorithm to the multiple dimensional space has been open so far. We extended the dynamic programming approach from one-dimensional clustering to multidimensional clustering with sequential constraint (i.e., only subsequent elements of the input may form a cluster). The method finds the exact optimum in this case as well. We implemented the algorithm in the R package **clustering.sc.dp** (Szkaliczki and Song, 2015).

Although the original algorithm has been developed to find the optimal solution with exactly k clusters it can determine the optimal value for all numbers of clusters less than or equal to k in a single run. For this reason, we implemented two variants of the algorithm. The first one finds the optimal solution for a specific k which can be used if the number of clusters is known in advance. The second variant returns the vector containing the minimal *withinss* for all cluster numbers less than or equal to k. This extension of the algorithm is useful if the number of clusters is not known in advance which is a common case.

The remainder of this paper is organized as follows: In the next section, a brief overview of the related work is presented. Then the optimization problem is formally described and the developed optimization algorithm is introduced in detail. Some evaluation results are also presented and the usage of the implemented package is introduced. A brief summary concludes the paper.

Related work

Several clustering models and a broad variety of clustering methods are available in the literature (Jain, 2010; Tan et al., 2006). Minimising *withinss* is a common optimisation goal used, e.g., in the

popular k-means method (Lloyd, 1982) to find a solution for a specific number of clusters and in the Ward's method (Ward, Jr., 1963) belonging to the hierarchical clustering methods. The problem is NP-complete (Aloise et al., 2009; Dasgupta and Freund, 2009; Mahajan et al., 2009). The general clustering methods cannot be directly applied to our problem because the produced solution usually violates the sequential constraint.

As mentioned, one-dimensional clustering can be solved in polynomial time (Wang and Song, 2011). The problem represents a special kind of clustering with sequential constraint because a necessary condition for the optimality in one dimension is that only subsequent items may form a cluster if the items are considered in their scalar order. The package presented in this paper generalised the one-dimensional clustering method to the multidimensional case. The dynamic programming method used for optimal clustering in one dimension is essentially the same as the one first applied by Bellman (1961) for linear curve approximation. For this reason, our package can be also considered as an implementation of the optimal dynamic programming clustering method proposed by Bellman.

Several papers (e.g., Himberg et al. 2001, Terzi 2006, Tierney et al. 2014) are dealing with clustering with sequential constraints because processing data sequences has a broad application area. Leiva and Vidal (2013) gave a clustering algorithm called Warped k-means for minimising *withinss* while considering the sequential constraint. The algorithm tries to reach the optimum by moving items between subsequent clusters. It does not guarantee optimality. Their paper provides a good overview as well on the taxonomy of the problem.

The problem specification

Clustering methods divide a dataset $X = \{\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\}$ of d-dimensional vectors into a set $\Pi = \{C_1, C_2, \ldots, C_k\}$ of disjoint clusters where n and k denote the number of items to be clustered and the number of clusters, respectively. Throughout the paper, vectors are distinguished from scalars by a bar over their symbol. In case of clustering with sequential constraint, the items are sorted and the clusters are formed only by subsequent items: $C_j = \left\{\overline{x}_{b_j}, \overline{x}_{b_j+1}, \ldots, \overline{x}_{b_j+n_j-1}\right\}$ where b_j and n_j denote the first item and the number of items in cluster C_j, respectively. The optimisation goal is to minimise the within-cluster sum of squared distances (*withinss*) also called sum of squared error (SSE), sum of quadratic errors (SQE) or distortion which is a common measurement of quality in clustering. It is formally defined as follows:

$$withinss = \sum_{j=1}^{k} \sum_{\overline{x}_i \in C_j} \left\| \overline{x}_i - \overline{\mu}_j \right\|^2, \tag{1}$$

where $\|\overline{x}\|$ denotes the Euclidean norm of vector $\overline{x}$ and $\overline{\mu}_j$ is the cluster mean:

$$\overline{\mu}_j = \frac{1}{n_j} \sum_{\overline{x}_i \in C_j} \overline{x}_i. \tag{2}$$

Now, we can formulate our problem as follows:

Input:

Items to be clustered: $X = \{\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\}$,

Number of clusters: k.

Output:

Optimal clustering: $\Pi = \{C_1, C_2, \ldots, C_k\}$.

Minimise

within-cluster sum of squared distances (*withinss*): Eq. (1).

Subject to

sequential constraint:

$$\left(\overline{x}_{i_1} \in C_j\right) \wedge \left(\overline{x}_{i_2} \in C_j\right) \wedge (i_1 \leq i_3 \leq i_2) \Rightarrow \left(\overline{x}_{i_3} \in C_j\right). \tag{3}$$

general clustering conditions:

each item is clustered:

$$\forall \overline{x}_i \exists C_j : \overline{x}_i \in C_j. \tag{4}$$

one cluster is assigned to each item

$$\left(\overline{x}_i \in C_{j_1}\right) \wedge \left(\overline{x}_i \in C_{j_2}\right) \Rightarrow (j_1 = j_2)\,. \tag{5}$$

The dynamic programming algorithm

The recursive formula used in the dynamic programming formulation is based on the fact, that if clustering for the first i items in m clusters is optimal then after dropping the last cluster the resulting clustering is optimal with $m-1$ clusters for the remaining items. Let $D[i,m]$ denote the value of the minimal within-cluster sum of squared distances (*withinss*) of the clustering for the first i items by using m clusters. If j denotes the first item of the mth cluster then the optimality of $D[i,m]$ implies the optimality of $D[j-1,m-1]$ as well. $D[n,k]$ gives the minimal *withinss* for clustering all items in k clusters where n denotes the total number of items.

The recursive formula applied in the dynamic programming approach is defined as follows (Wang and Song, 2011):

$$D[i,m] = \min_{m \le j \le i} \left\{ D[j-1,m-1] + d\left(\overline{x}_j, \ldots, \overline{x}_i\right) \right\}, 1 \le i \le n, 1 \le m \le k \tag{6}$$

where $d\left(\overline{x}_j, \ldots \overline{x}_i\right)$ is the sum of squared Euclidean distances from $\overline{x}_j, \ldots \overline{x}_i$ to their mean.

The optimal solution can be determined by dynamic programming in two steps. First, the recursive formula is used to find the minimal *withinss*. Then backtracking finds the optimal clustering.

$B[i,m]$ stores the index of the first item b_m of the last cluster in the partial solution belonging to $D[i,m]$ which is used for backtracking the optimal solution after determining the minimal *withinss*. We apply the dynamic programming method to solve optimal clustering for a range of cluster numbers and k denotes the maximum number of clusters in our algorithm ($1 \le k \le n$). The steps of calculating $D[i,m]$ can be implemented as follows:

```
for i := 0 to n
  D[i, 0] := 0        // initialisations
for i := 1 to n
  for m := 1 to min(i, k)
    D[i, m] := MAX_DOUBLE;
    for j := i downto m        // calculating the recursive formula
      if D[i, m] > D[j - 1, m - 1] + d(xj, ..., xi)
        D[i, m] := D[j - 1, m - 1] + d(xj, ..., xi)
        B[i, m] := j
Return D[n, m] for m = 1, ..., k
```

$D[n,m], m = 1, \ldots, k$ gives the minimal distances for different number of clusters. If the number of clusters is known in advance, it is enough to return $D[n,k]$. Otherwise, $D[n,m]$ for all $m \le k$ can be returned for further processing by the user in order to select the proper number of clusters.

The algorithm finds the exact optimum in polynomial time. It runs $O\left(n^2 k\right)$ iterations in which $D[i,m]$ is checked and, if necessary, updated. Each iteration can be performed in time proportional to the dimensions of the vectors ($O(d)$) independently from the number of items and clusters if $d\left(\overline{x}_j, \ldots, \overline{x}_i\right)$ is computed progressively based on $d\left(\overline{x}_{j+1}, \ldots, \overline{x}_i\right)$ and the average of the items. This can be done similarly to the one-dimensional case in the following way. Let $\overline{\mu}_{j,i}$ denote the mean of the items with index between j and i. If $j = i$ $d\left(\overline{x}_j, \ldots, \overline{x}_i\right) = 0, \overline{\mu}_{j,i} = \overline{x}_i$. For index j from $i-1$ down to m, the algorithm iteratively computes

$$d\left(\overline{x}_j, \ldots, \overline{x}_i\right) = d\left(\overline{x}_{j+1}, \ldots, \overline{x}_i\right) + \frac{i-j}{i-j+1}\left(\overline{x}_j - \overline{\mu}_{j,i-1}\right)^2$$

$$\overline{\mu}_{j,i} = \frac{\overline{x}_j + (i-j)\,\overline{\mu}_{j+1,i}}{i-j+1}$$

Using the above iterative computation, the overall running time is quadratic in the number of items and linear in the number of clusters and the dimensions of the vectors: $O\left(n^2 k d\right)$.

The optimal solution with cluster number m can be backtracked by the help of $B[i,m]$ (Wang and Song, 2011):

$$B[i,m] = \text{argmin}_{m \le j \le i} \left\{ D[j-1,m-1] + d\left(\overline{x}_j, \ldots, \overline{x}_i\right) \right\}, 1 \le i \le n, 1 \le m \le k \tag{7}$$

$B[n, m]$ is equal to the first item of the last cluster in clustering with m clusters. The last cluster contains items $\overline{x}_{B[n,m]}, \ldots, \overline{x}_n$. The further clusters can be determined by using backtracking as follows: if j is the first item of the lth cluster then the preceding cluster is formed by the items $\overline{x}_{B[j,l-1]}, \ldots, \overline{x}_{j-1}$. The steps of backtracking to the find optimal clustering using m number of clusters are as follows:

```
the mth cluster is B[n, m], ..., n
j := B[n, m]
for l := m to 2
  the (l - 1)th cluster is B[j, l - 1], ..., j - 1
  j = B[j, l - 1]
```

Backtracking can be performed in linear time in the number of the clusters ($O(k)$).

We would like to mention how to combine the dynamic programming method with methods finding the proper number of clusters. The cluster numbers are typically determined by using a measure of validity (e.g., *withinss*) indicating the goodness of clustering. The "best" k is chosen based on the analysis of the values of the measure for each k within the range of the possible number of clusters. A huge variety of methods are available in the literature to determine the cluster numbers (Milligan and Cooper, 1985; Dimitriadou et al., 2002). Typically, they are applied on the output generated by hierarchical clustering methods. Although our dynamic programming approach does not belong to the hierarchical clustering similar methods can be used on the result of our algorithm for finding the proper number of clusters.

Backtracking can be performed only once if the number of clusters is known in advance or it can be selected by analysing *withinss* contained in the last column of matrix D. Otherwise, backtracking should be executed for each possible cluster numbers for further analysis. The method can efficiently determine the optimal clustering for all numbers of clusters less than or equal to k essentially because the most time-consuming part is determining D and B which should be executed only once.

Implementation

We implemented this dynamic programming algorithm in C++. The implementation was built on source code from the R package **Ckmeans.1d.dp** and we created a new R package **clustering.sc.dp**. In the name of the package, *sc* and *dp* refer to sequential constraint and dynamic programming, respectively. The open-source approach made it possible to reuse the code from package **Ckmeans.1d.dp** but it was beneficial for the original package as well: we suggested a minor change in the code to speed up the code which was incorporated into **Ckmeans.1d.dp** ($\geq$ version 3.3.0).

Evaluation

Optimality

If the sequential constraint is considered in clustering than the optimal *withinss* is usually larger than in the general case without the constraint since the constraint excludes many possible solutions. In order to compare our algorithm with general clustering methods such as the k-means method, we generated a dataset where the optima without and with sequential constraint are equal. For this purpose, we created a totally ordered vector set where one vector is simultaneously larger or smaller in each coordinate than another vector ($\forall i, j \in \{1, 2, \ldots, n\} \forall k, l \in \{1, 2, \ldots, d\} x_{ik} < x_{jk} \implies x_{il} < x_{jl}$ where x_{yz} denote the zth coordinate of item x_y). We used a random walk as a totally ordered vector set where the steps between subsequent items were generated by using the exponential distribution. The generated dataset consisted of 10,000 two-dimensional vectors.

We compared the dynamic programming algorithm with the `kmeans()` function in R which provides the Hatigan and Wong (1979) implementation of k-means. We ran the algorithms with different cluster numbers from 2 to 50. The minimal *withinss* found by k-means was always greater or equal to the optimum value found by `clustering.sc.dp()`.

We used the relative difference in *withinss* from the `kmeans()` result to the optimal value produced by `clustering.sc.dp()` for measuring deviation of the k-means result from the optimum. Figure 1 shows the relative difference as a function of the cluster numbers. It can be seen that k-means is able to find the optimum if the cluster number is at most 10. For larger cluster numbers, its error starts increasing and the relative difference is more than 20% if the cluster number is 50.

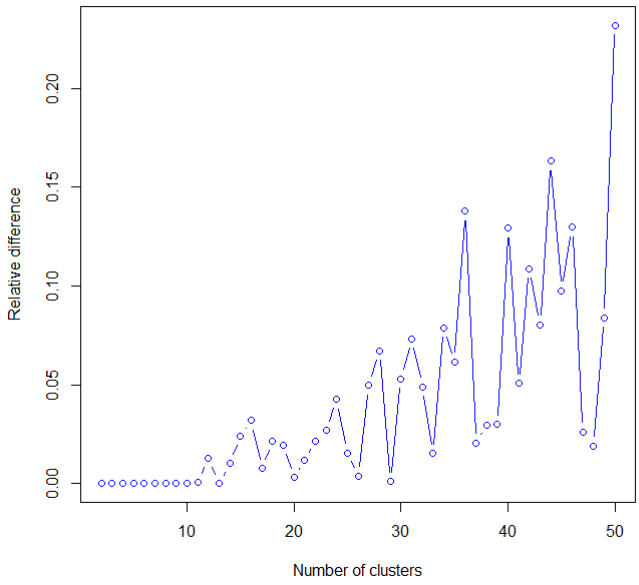


Figure 1: The relative difference in *withinss* from `kmeans()` to the optimal value returned by `clustering.sc.dp()`. The input data set of size 10 000 were generated as a random walk having a step size that varies according to an exponential distribution with rate 1.0 in each coordinate.

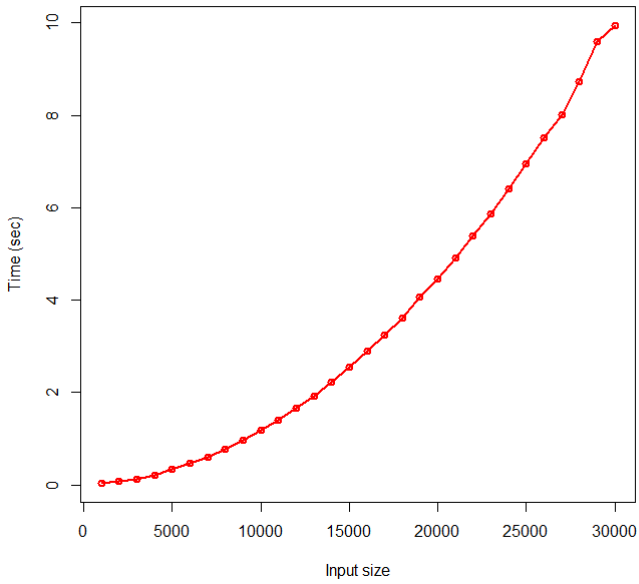


Figure 2: Runtime as a function of the number of the items to be clustered.

Runtime

We tested the dynamic programming algorithm on inputs with different sizes, dimensions and cluster numbers in order to find its performance bounds and examine experimentally how the running time depends on the input sizes. The simulations were run on a desktop computer with a Pentium Dual-Core 2.93 GHz processor and 4 GB memory, running Windows 10 operation system. We generated multidimensional Gaussian random walks as data sets for performance tests. The steps between subsequent items were generated independently for each coordinate by using the Gaussian random distribution with zero mean and standard deviation of 0.1.

In the first setting (Figure 2), runtime is obtained as a function of input data size for running `clustering.sc.dp()`. The size of the input varies from 1,000 to 30,000 with a step size of 1,000. The input data consists of two-dimensional vectors. The number of clusters is set to 2. The runtime increases quadratically in the number of items to be clustered.

Table 1 presents some runtime data for a different magnitude of the number of items in order to

Size	1000	10,000	100,000	1,000,000
Runtime (sec)	0.03	1.19	144.00	14,479.88

Table 1: Runtime for different numbers of the items to be clustered.

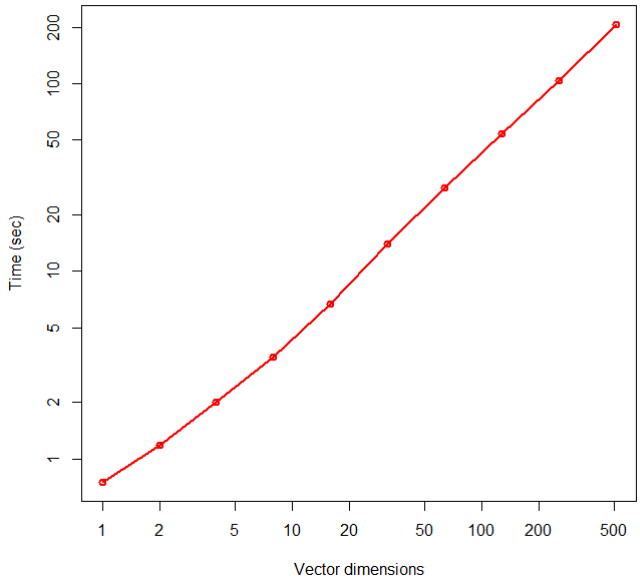


Figure 3: Runtime as a function of the dimension of the items to be clustered.

find performance bounds of the method. One can see that the optimal clustering was found very quickly if the number of items is 1,000, it took more than a second, almost two and a half minutes, about four hours for 10,000, 100,000 and one million items, respectively.

In the second performance test we examine the dependency of the runtime on the dimensions. All input data sets are of the same size 10,000 and the dimensions of the vectors varies from 1 to 512. The dimensions are doubled in each run. Figure 3 shows the results. The algorithm runs less than a second if the input contains one-dimensional vectors (i.e., scalars). The runtime increases linearly with the dimension and it can be solved within two and a half minutes if the dimension of the processed vectors is 512.

In the third performance test, the runtime is examined as a function of the number of clusters. The number of clusters is between 1 and 25. The input data set consist of 10,000 two-dimensional vectors. The black line with circles in Figure 4 shows the result. The runtime increases linearly with the number of clusters.

Finally, we compared the runtime of different functions within the package. The input data were the same as in the previous setting. One can see in Figure 4 that the runtime of `findwithinss.sc.dp()` is slightly larger than the one of `clustering.sc.dp()`. The runtime of `backtracking.sc.dp()` remains small even for large cluster numbers. The package has three typical ways of usage:

- `clustering.sc.dp()` can be called when the number of clusters is known in advance.
- `findwithinss.sc.dp()` can be called first and then `backtracking.sc.dp()` for a selected number of clusters.
- `findwithinss.sc.dp()` can be called first and then `backtracking.sc.dp()` for all possible number of clusters.

The subsequent call of `findwithinss.sc.dp()` and `backtracking.sc.dp()` is only slightly slower than calling `clustering.sc.dp()`. Furthermore, the total runtime of calling `findwithinss.sc.dp()` for cluster number k and `backtracking.sc.dp()` for all cluster numbers less than or equal to k is less than the double of the runtime of `clustering.sc.dp()` called for the same cluster number. This is much faster than calling the clustering function k times which would be required without splitting `clustering.sc.dp()` into phases of finding the optimal *withinss* (`findwithinss.sc.dp()`) and backtracking (`backtracking.sc.dp()`).

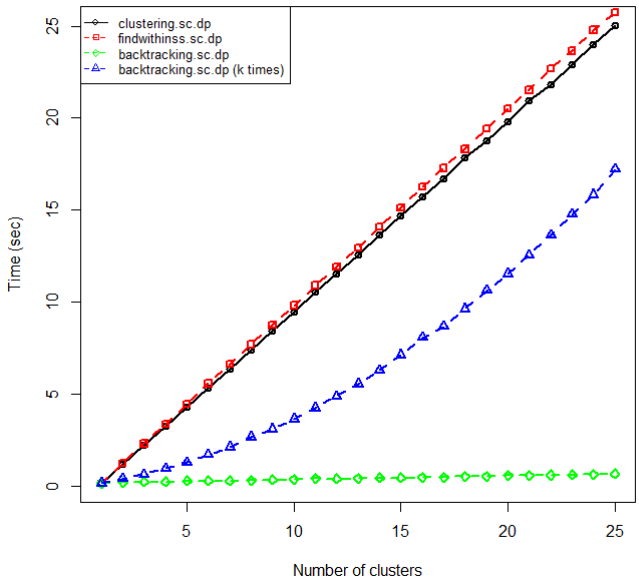


Figure 4: Comparison of runtime as a function of number of clusters between different functions provided by package **clustering.sc.dp**: `clustering.sc.dp()`, `findwithinss.sc.dp()`, `backtracking.sc.dp()`. It also contains the total running time of `backtracking.sc.dp()` when it is called for all cluster numbers less than or equal to the specific cluster number.

Introduction to the R package clustering.sc.dp

R package **clustering.sc.dp** offers functions to perform optimal clustering on multidimensional data with sequential constraint. Method `clustering.sc.dp()` can find the optimal clustering if the number of clusters is known. Otherwise, methods `findwithinss.sc.dp()` and `backtracking.sc.dp()` can be used.

The following examples illustrate how to use the package. Function `clustering.sc.dp()` outputs the same fields describing clustering as `Ckmeans.1d.dp()` does in the original package:

- *cluster*: a vector of cluster indices assigned to each element in x. Each cluster is indexed by an integer from 1 to k.
- *centers*: a matrix whose rows represent the vectors of cluster centres (the average of the points within the cluster).
- *withinss*: the within-cluster sum of squared distances for each cluster.
- *size*: a vector containing the number of points in each cluster.

Figure 5 visualizes the input data and the clusters created by `clustering.sc.dp()`. See below for the R source code of the example.

```
# Example1: clustering data generated from a random walk
x <- rbind(0, matrix(rnorm(99 * 2, 0, 0.1), nrow = 99, ncol = 2))
x <- apply(x, 2, cumsum)

k <- 2
result <- clustering.sc.dp(x, k)
plot(x, type = "b", col = result$cluster)
points(result$centers, pch = 24, bg = 1:k)
```

The next example demonstrates the usage of functions `findwithinss.sc.dp()` and `backtracking.sc.dp()`. Similarly to the previous example, it also processes data of a random walk. Function `findwithinss.sc.dp()` finds optimal *withinss* for a range of cluster numbers. It returns a list with two components:

- *withinss*: a vector of total within-cluster sum of squared distances of the optimal clusterings for each number of clusters less than or equal to `k`.
- *backtrack*: backtrack data used by `backtracking.sc.dp()`.

In our example, the first cluster number where *withinss* drops below a threshold is selected as the number of clusters. Function `backtracking.sc.dp()` outputs the optimal clustering for the selected

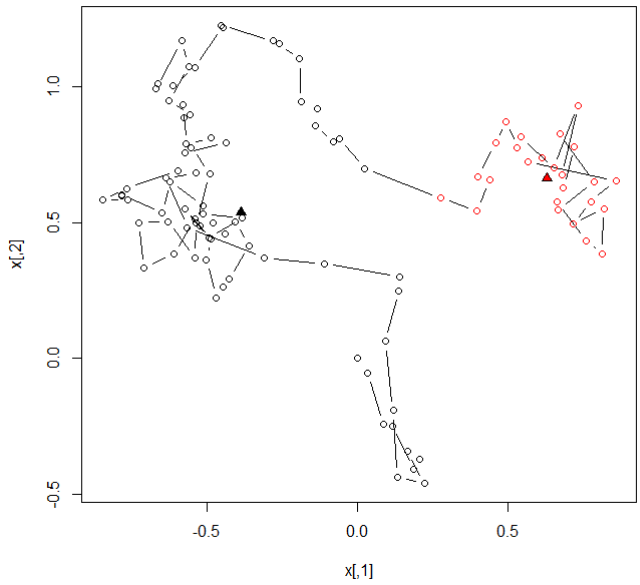


Figure 5: Clustering data representing a two-dimensional random walk into two clusters. The points to be clustered are represented by circles. The clusters are indicated using different colours. The cluster centres are denoted by triangles.

cluster number in the same format as `clustering.sc.dp()` does without running the whole clustering process again (Figure 6).

```
# Example2: clustering data generated from a random walk with small withinss
x <- rbind(0, matrix(rnorm(99 * 2, 0, 0.1), nrow = 99, ncol = 2))
x <- apply(x, 2, cumsum)

k <- 10
r <- findwithinss.sc.dp(x, k)

# select the first cluster number where withinss drops below a threshold
k_th <- which(r$twithinss <= 5.0)[1]

# backtrack
result <- backtracking.sc.dp(x, k_th, r$backtrack)
plot(x, type = "b", col = result$cluster)
points(result$centers, pch = 24, bg = 1:k_th)
```

Summary

Clustering data with sequential constraint is a polynomial time solvable variant of the clustering problem. The paper introduced a package implementing a dynamic programming approach that finds the exact optimum of the problem. The algorithm represents an extension of the one-dimensional dynamic programming strategy of **Ckmeans.1d.dp** to multiple dimensional spaces which has been an open problem in the paper of Wang and Song (2011). The package supports both cases when the exact number of clusters is given and when the number of clusters is not known in advance. It can also be used to evaluate approximation algorithms for clustering with sequential constraint due to its optimality. The runtime evaluations indicate how fast the algorithm can solve problems with different sizes and parameters. Our future plan is to use the dynamic programming method in video summarisation.

Acknowledgements

Research is supported by the Hungarian National Development Agency under grant HUMAN_MB08-1-2011-0010.

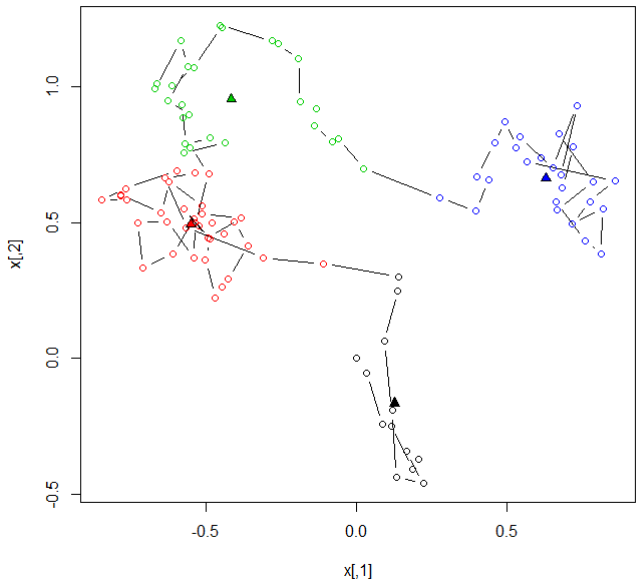


Figure 6: Clustering data representing a two-dimensional random walk. The number of clusters was determined by the analysis of optimal *withinss* for a range of cluster numbers. It uses the same notation as Figure 5.

Bibliography

D. Aloise, A. Deshpande, P. Hansen, and P. Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, Jan. 2009. [p319]

R. Bellman. On the approximation of curves by line segments using dynamic programming. *Communication of the ACM*, 6(6):284, 1961. [p319]

S. Dasgupta and Y. Freund. Random projection trees for vector quantization. *IEEE Transactions on Information Theory*, 55(7):3229–3242, July 2009. [p319]

E. Dimitriadou, S. Dolnicar, and A. Weingessel. An examination of indexes for determining the number of clusters in binary data sets. *Psychometrika*, 67(1):137–160, 2002. [p321]

J. A. Hatigan and M. A. Wong. Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society C*, 28(1):100–108, 1979. [p321]

J. Himberg, K. Korpiaho, H. Mannila, J. Tikanmaki, and H. Toivonen. Time series segmentation for context recognition in mobile devices. In *Proceedings of the IEEE International Conference on Data Mining, 2001 (ICDM 2001)*, pages 203–210, 2001. [p319]

A. K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010. [p318]

L. A. Leiva and E. Vidal. Warped k-means: An algorithm to cluster sequentially-distributed data. *Information Sciences*, 237:196–210, July 2013. [p319]

S. Lloyd. Least squares quantization in PCM. *IEEE Transations on Information Theory*, 28(2):129–137, 1982. [p319]

M. Mahajan, P. Nimbhorkar, and K. Varadarajan. The planar k-means problem is NP-hard. In *Proceedings of the 3rd International Workshop on Algorithms and Computation (WALCOM '09)*, pages 274–285, Berlin, Heidelberg, 2009. Springer-Verlag. [p319]

G. W. Milligan and M. C. Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50(2):159–179, 1985. [p321]

M. Song and H. Wang. Ckmeans.1d.dp: Optimal k-means clustering for one-dimensional data. *R package version 3.02*, 2011. URL http://CRAN.R-project.org/package=Ckmeans.1d.dp. [p318]

T. Szkaliczki and J. Song. clustering.sc.dp: Optimal distance-based clustering for multidimensional data with sequential constraint. *R package version 1.0*, 2015. URL http://CRAN.R-project.org/package=clustering.sc.dp. [p318]

P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison Wesley, 2006. [p318]

E. Terzi. Efficient algorithms for sequence segmentation. In *Proceedings of the Sixth SIAM International Conference on Data Mining*, pages 314–325, 2006. [p319]

S. Tierney, J. Gao, and Y. Guo. Subspace clustering for sequential data. In *Proceedings of the IEEE Computer Conference on Computer Vision and Pattern Recognition*, pages 1019–1026, 2014. [p319]

H. Wang and M. Song. Ckmeans.1d.dp: Optimal k-means clustering in one dimension by dynamic programming. *The R Journal*, 3(2):29–33, 2011. [p318, 319, 320, 325]

J. H. Ward, Jr. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963. [p319]

Tibor Szkaliczki
eLearning Department
Institute for Computer Science and Control, Hungarian Academy of Sciences
Hungary
szkaliczki.tibor@sztaki.mta.hu

progenyClust: an R package for Progeny Clustering

by Chenyue W. Hu and Amina A. Qutub

Abstract Identifying the optimal number of clusters is a common problem faced by data scientists in various research fields and industry applications. Though many clustering evaluation techniques have been developed to solve this problem, the recently developed algorithm *Progeny Clustering* is a much faster alternative and one that is relevant to biomedical applications. In this paper, we introduce an R package **progenyClust** that implements and extends the original *Progeny Clustering* algorithm for evaluating clustering stability and identifying the optimal cluster number. We illustrate its applicability using two examples: a simulated test dataset for proof-of-concept, and a cell imaging dataset for demonstrating its application potential in biomedical research. The **progenyClust** package is versatile in that it offers great flexibility for picking methods and tuning parameters. In addition, the default parameter setting as well as the plot and summary methods offered in the package make the application of *Progeny Clustering* straightforward and coherent.

Introduction

Clustering is a classical and widely-used machine learning technique, yet the field of clustering is constantly growing. The goal of clustering is to group objects that are similar to each other and separate objects that are not similar to each other based on common features. Clustering can, for example, be applied to distinguishing tumor subclasses based on gene expression data (Sørlie et al., 2001; Budinska et al., 2013), or dividing sport fans based on their demographic information (Ross, 2007). One critical challenge in clustering is identifying the optimal number of groups. Despite some advanced clustering algorithms that can automatically determine the cluster number (e.g. Affinity Propagation (Frey and Dueck, 2007)), the commonly used algorithms (e.g. k-means (Hartigan and Wong, 1979) and hierarchical clustering (Johnson, 1967)) unfortunately require users to specify the cluster number before performing the clustering task. However, most often than not, the users do not have prior knowledge of the number of clusters that exist in their data.

To solve this challenge of finding the optimal cluster number, quite a few clustering evaluation techniques (Arbelaitz et al., 2013; Charrad et al., 2014a) as well as R packages (e.g. **cclust** (Dimitriadou et al., 2015), **clusterSim** (Walesiak et al., 2015), **cluster** (Maechler et al., 2015), **Nbclust** (Charrad et al., 2014b), **fpc** (Hennig, 2015)) were developed over the years to objectively assess the clustering quality. The problem of identifying the optimal cluster number is thus transformed into the problem of clustering evaluation. In most of these solutions, clustering is first performed on the data with each of the candidate cluster numbers. The quality of these clustering results is then evaluated based on properties such as cluster compactness (Tibshirani et al., 2001; Rousseeuw, 1987) or clustering stability (Ben-Hur et al., 2001; Monti et al., 2003). In particular, stability-based methods have been well received and greatly promoted in recent years (Meinshausen and Bühlmann, 2010). However, these methods are generally slow to compute because of the repetitive clustering process mandated by the nature of stability assessment. Recently, a new method *Progeny Clustering* was developed by Hu et al. (2015) to assess clustering quality and to identify the optimal cluster number based on clustering stability. Compared to other clustering evaluation methods, *Progeny Clustering* requires fewer samples for clustering stability assessment, thus it is able to greatly boost computing efficiency. However, this advantage is based on the assumption that features are independent for each cluster, thus users need to either transform data and create independent features or consult other methods if this assumption does not hold for the data of interest.

Here, we introduce a new R package, **progenyClust**, that performs *Progeny Clustering* for continuous data. The package consists of a main function `progenyClust()` that requires few parameter specifications to run the algorithm on any given dataset, as well as a built-in function `hclust.progenyClust` to use hierarchical clustering as an alternative to using `kmeans`. Two example datasets `test` and `cell`, used in the original publication of *Progeny Clustering*, are provided in this package for testing and sharing purposes. In addition, the **progenyClust** package includes an option to invert the stability scores, which is not considered in the original algorithm. This additional capability enables the algorithm to produce more interpretable and easier-to-plot results. The rest of the paper is organized as follows: We will first describe how *Progeny Clustering* works and then go over the implementation of the **progenyClust** package. Following the description of functions and datasets provided by the package, we will provide one proof-of-concept example of how the package works and a real world example where the package is used to identify cell phenotypes based on imaging data.

Progeny Clustering

In this section, we briefly review the algorithm of *Progeny Clustering* (Hu et al., 2015). *Progeny Clustering* is a clustering evaluation method, thus it needs to couple with a stand-alone clustering method such as *k-means*. The framework of *Progeny Clustering* is similar to other stability based methods, which select the optimal cluster number that renders the most stable clustering. The evaluation of clustering stability usually starts with an initial clustering of the full or sometimes partial dataset, followed by bootstrapping and repetitive clustering, and then uses certain criterion to assess the stability of clustering solutions. *Progeny Clustering* uses the same workflow, but innovates at the bootstrapping method and improves on the stability assessment.

Consider a finite dataset $\{x_{ij}\}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$ that contains M variables (or features) for N independent observations (or samples). Given a number K (a positive integer) for clustering, a clustering method partitions the dataset into K clusters. Each cluster is denoted as C_k, $k = 1, \ldots, K$. Inspired by biological concepts, each cluster is treated as a subpopulation and the bootstrapped samples as progenies from that subpopulation. The uniqueness of *Progeny Sampling* during the bootstrapping step is that it randomly samples feature values with replacement to construct new samples rather than directly sampling existing samples. Let $\tilde{N}$ be the number of progenies we generate from each cluster C_k. Combining these progenies, we have a validation dataset $\{y_{ij}^{(k)}\}$, $i = 1, \ldots, \tilde{N}$, $j = 1, \ldots, M$, $k = 1, \ldots, K$, containing $K \times \tilde{N}$ observations with M features. Using the same number K and the same method for clustering, we partition the progenies $\{y_{ij}^{(k)}\}$ into K progeny clusters, denoted by C'_k, $k = 1, \ldots, K$. A symmetric co-occurrence matrix Q records the clustering memberships of each progeny as follows:

$$Q_{ab} = \begin{cases} 1, & \text{if the } a^{th} \text{ progeny and the } b^{th} \text{ progeny are in the same cluster } C'_k \\ 0, & \text{otherwise} \end{cases}. \tag{1}$$

The progenies in Q were ordered by the initial cluster (C_k) they were generated from, such that $Q_a, \ldots, Q_{a+\tilde{N}} \in C_k$, $a = (k-1)\,\tilde{N}$. After repeating the above process (from generating *Progenies* to obtaining Q) R times, we can get a series of co-occurrence matrices $Q^{(r)}$, $r = 1, \ldots, R$. Averaging $Q^{(r)}$ results in a stability probability matrix P, i.e.

$$P_{ab} = \sum_r Q_{ab}^{(r)} / R \ . \tag{2}$$

From this probability matrix P, we compute the stability score for clustering the dataset $\{x_{ij}\}$ into K clusters as

$$S = \frac{\sum_k \sum_{a,b \in C_k, b \neq a} P_{ab} / \left(\tilde{N} - 1\right)}{\sum_k \sum_{a \in C_k, b \notin C_k} P_{ab} / \left(K\tilde{N} - \tilde{N}\right)}. \tag{3}$$

A schematic for this process and the pseudocode are shown in Figure 1 and Figure 2.

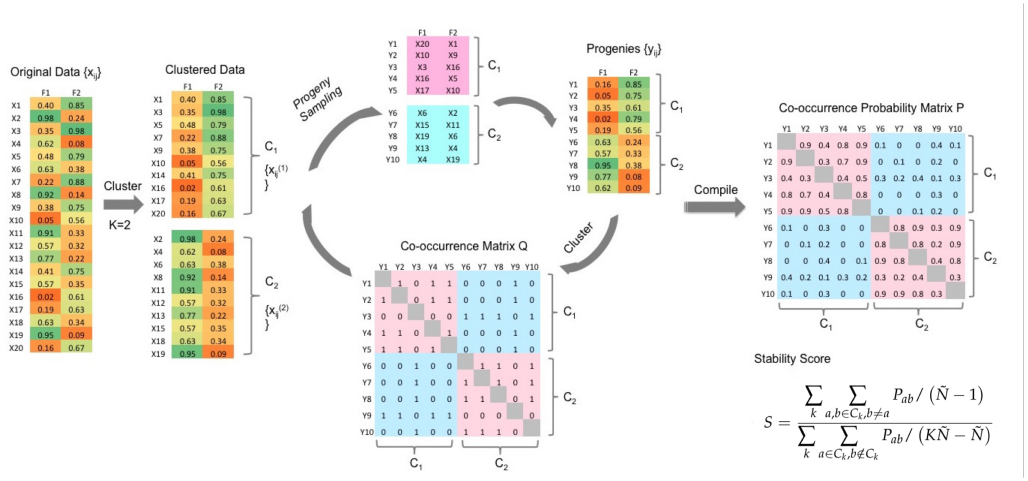


Figure 1: The schematic of the core steps in *Progeny Clustering*, illustrated using an example of clustering a 20×2 matrix into two groups. Schematic reproduced from Hu et al. (2015) under a Creative Commons License.

Input $\{x_{ij}\}$, K_{min}, K_{max}, $\tilde{N}$, R
for $K = K_{min}$ to K_{max} **do**
  Cluster $\{x_{ij}\}$ into K clusters
  Initiation Q = Zero Matrix of size $N \times N$
  for $r = 1$ to R **do**
    for $k = 1$ to K **do**
      Construct $\{y_{ij}^{(k)}\}$ of size $\tilde{N}$ by *Progeny Sampling* from $\{x_{ij}^{(k)}\}$
    end for
    Cluster $\{y_{ij}\}$ into K clusters
    $Q = Q + Q^{(r)}$
  end for
  $P^{(K)} = Q/R$
  Compute $\{S^{(K)}\}$ by Equation 3
end for
Output $\{S^{(K)}\}$

Figure 2: The pseudo code of the *Progeny Clustering* algorithm, from Hu et al. (2015).

After computing the stability score for each cluster number, we can then pick the optimal number using a 'greatest score' criterion or a 'greatest gap' criterion or both. The 'greatest score' criterion selects the cluster number that produces the highest stability score compared to reference datasets, similar to what is used in *Gap Statistics* (Tibshirani et al., 2001). T reference datasets are first generated from a uniform distribution over the range of each feature using Monte Carlo simulation. Each reference dataset is then treated as an input dataset, and stability scores are computed respectively using the same process as in Figure 1. Let $\{\tilde{S}^{(K)(t)}\}$, $t = 1, \ldots, T$, be the stability score for clustering the t^{th} reference dataset into K clusters. The stability score difference between the original dataset and reference datasets are obtained by

$$D^{(K)} = S^{(K)} - \sum_t \tilde{S}^{(K)(t)}/T, \tag{4}$$

where $K = K_{min}, \ldots, K_{max}$. The optimal cluster number with the greatest score difference is then selected, i.e.

$$K_o = \arg\max D^{(K)}. \tag{5}$$

While the 'greatest score' criterion requires computing stability scores from random datasets, the 'greatest gap' criterion does not, due to the fact that the stability score linearly increases with an increase in cluster number among reference datasets. The 'greatest gap' criterion therefore searches for peaks in the stability score curve and selects the cluster number that has the highest stability score compared to those of its neighboring numbers, i.e.

$$K_o = \arg\max \left(2S^{(K)} - S^{(K-1)} - S^{(K+1)}\right). \tag{6}$$

Compared to other stability-based evaluation methods, the major benefits of using *Progeny Clustering* include less re-use of the same samples and faster computation. The progenies sampled from the original data resemble but are hardly the same as the original samples. Thanks to this unique feature, a small number of progenies are sufficient to evaluate the clustering stability. The reduction of sample size for evaluation in turn saves substantial computing time, because the complexity of most clustering algorithms is dependent on the sample size (Andreopoulos et al., 2009). The proposal of the 'greatest gap' criterion further boosts computation speed of clustering evaluation by eliminating the step of generating reference scores. The comparison of computation speed between *Progeny Clustering* and other commonly used algorithms can be found in Hu et al. (2015).

The progenyClust package

The **progenyClust** package was developed with the aim of enabling and promoting the usage of the *Progeny Clustering* algorithm in the R community. This package implements the *Progeny Clustering* algorithm with an additional feature to invert stability scores. The package includes a main function `progenyClust()`, plot and summary methods for "progenyClust" objects, a function

`hclust.progenyClust` for hierarchical clustering, and two example datasets. To perform *Progeny Clustering* using the **progenyClust** package, users should first run the main function `progenyClust()` on their dataset, then use plot and summary methods to check the stability score curves, review the clustering results, and check the recommended cluster number. The `progenyClust()` function allows flexible plug-ins of various clustering algorithms into *Progeny Clustering*, and directly couples with *k-means* clustering algorithm as a default as well as hierarchical clustering as an alternative. Since the clustering memberships are returned in addition to the optimal cluster number, the package integrates the clustering process and the cluster number selection process into one, and it saves users additional efforts that are required to complete clustering tasks. In the following sections, we will first explain the motivation to provide score inversion, then go over the main `progenyClust()` function, the "S3" methods for "progenyClust" objects, the built-in function `hclust.progenyClust()`, and describe the background of the included datasets.

Inversion of the stability scores

In the original *Progeny Clustering* algorithm, the optimal cluster number was chosen based on stability scores, which capture the true classification rate over the false classification rate. The higher the score is, the more stable the clustering is, and the more desirable the cluster number is. The computation of stability scores works well in general, except for when the false classification rate is equal to zero. The zero false classification rate indicates a perfectly stable clustering, that is when all progenies are correctly clustered with progenies coming from the same initial cluster. The perfectly stable clustering will produce a positive infinite stability score, which is not ideal for plotting or for further computing to select the optimal cluster number. Therefore, we offer a choice of inverting the stability scores in this package to mitigate the risk of generating an infinite score. The inverted stability scores can be interpreted as a measure of instability, calculated by false classification rate over true classification rate. In the case of a perfectly stable clustering, the inverted stability score is equal to zero, thus is much easier for comparison and visualization. Meanwhile, the chances of a perfectly unstable clustering are much rarer. If the inversion of stability score is chosen when running *Progeny Clustering*, users should select the cluster number with the smallest score instead of the greatest score.

The `progenyClust()` function

The `progenyClust()` function takes in a data matrix, performs *Progeny Clustering*, and outputs a "progenyClust" object. The clustering is performed on rows, thus the input data matrix needs to be formatted accordingly. A number of input arguments were offered by `progenyClust()` to allow users to specify the clustering algorithm, cluster number selection criterion and parameter values they want to use for *Progeny Clustering*. The output "progenyClust" object contains information on the clustering memberships and stability scores at each cluster number, and it can work with the `plot` and `summary` methods. Since the default values for most of the input arguments are provided, `progenyClust()` can be run without any tuning. The function is used as follows:

```
progenyClust(data, FUNclust = kmeans, method = 'gap', score.invert = F,
        ncluster = 2:10, size = 10, iteration = 100, repeats = 1, nrandom = 10, ...)
```

Here, we group the input arguments into three categories, and highlight the meaning and usage of each argument.

- **Input Data:** `data` is a matrix, the rows of which are of interest to cluster. `ncluster` is a sequence of candidate cluster numbers to evaluate.
- **Method:** Since `progenyClust()` is a clustering evaluation algorithm, it needs to work together with a clustering algorithm. `FUNclust` is where the clustering function is specified. The input and output of `FUNclust` is required to be similar to the default `kmeans()` function from **stat**, or the alternative `hclust.progenyClust()` function for hierarchical clustering as provided in **progenyClust**. `FUNclust` should be able to accept `data` as its first argument, accept the number for clustering as its second argument, and return a list containing a component `cluster` which is a vector of integers denoting the clustering assignment for each sample. `method` is the stability score comparison criterion being selected. `score.invert` can be used to flip the stability scores to instability scores when specified to be `TRUE`. The values of `method` can be 'gap' which represents the 'greatest gap' criterion, 'score' which represents the 'greatest score' criterion, or 'both' which represents using both the 'greatest gap' and the 'greatest score' criteria. In cases when optimal cluster numbers determined by the 'greatest gap' and the 'greatest score' do not agree, we suggest users to either review the stability score plots from both criteria and pick the most preferred one or use the cluster number suggested by the 'greatest score' criterion.

- **Tuning Parameters:** `size` specifies the number of progenies to generate from each initial cluster for stability evaluation. `iteration` denotes how many times the progenies are generated for calculating the stability score. `repeats` is the number of times the entire algorithm should be repeated from the initial clustering to obtaining the stability scores. If `repeats` is greater than one, the standard deviation of the stability score at each cluster number will be produced. `nrandom` specifies the number of random datasets to generate when computing the reference scores, if the 'greatest score' method is chosen. All these tuning parameters, if specified inappropriately, can affect the accuracy and computing efficiency of the *Progeny Clustering* algorithm. In general, the greater the values of `size`, `iteration`, `repeats` and `nrandom` are, the slower the computing will be.

The output of the `progenyClust()` function is an object of the "progenyClust" class, which contains information on the clustering results, the stability scores computed and the calls that were made. Specifically, `cluster` is a matrix of clustering memberships, where rows are samples and columns are cluster numbers; `mean.gap` and `mean.score` are the scores computed at each given cluster number and normalized based on the 'greatest gap' and the 'greatest score' criteria; `score` and `random.score` are the initial stability scores computed before using any criteria to normalize; `sd.gap` and `sd.score` are the standard deviations of the scores when the input argument `repeats` is specified to be greater than one; `call`, `ncluster`, `method` and `score.invert` return the call that was made and input arguments specified.

The `plot` and `summary` methods for "progenyClust" objects

To identify the optimal cluster number, we provide the S3 `plot` and `summary` methods for "progenyClust" objects. The `plot` method enables users to visualize stability scores for cluster number selection and to visualize the clustering results. The `plot` function is as follows:

```
plot(x, data = NULL, k = NULL, errorbar = FALSE, xlab = '', ylab = '', ...)
```

If `data` is not provided, the function will visualize the stability score at each investigated cluster number to give users an overview of the clustering stability. When `data` is provided, the function will visualize `data` in scatter plots and represent each cluster membership by a distinct color. `data` can be the orginal data matrix used for clustering or a subset of the original data with fewer variables but the same number of samples. Additional graphical arguments can be passed to customize the plot. The only extra input argument we added here is `errorbar`, which will render error bars when plotting stability scores if `errorbar = TRUE`. The `errbar` function from **Hmisc** (Harrell Jr and Harrell Jr, 2015) was used to generate the error bars. In addition, the `summary` method of the "progenyClust" object produces a quick summary of what number of clusters is the best to use for the given data.

The `hclust.progenyClust()` function

The `hclust.progenyClust()` function performs hierarchical clustering by combining three existing R functions `dist()`, `hclust()` and `cutree()` from **stat** into one. The input and output are formatted such that they can be directly plugged into the `progenyClust()` function as an option for `FUNclust`, similar to the default `kmeans()` function. The function is as follows:

```
hclust.progenyClust(x, k, h.method = 'ward.D2', dist = 'euclidean', p = 2, ...)
```

To ensure consistency between similar R functions and allow users to easily use this function, the input arguments are largely kept the same as the ones used in fucntions `dist()`, `hclust()`, `cutree()`. The function returns clustering memberships, an `hclust` object of the tree, and a `dist` object of the distance matrix.

The `test` and `cell` datasets

A couple of datasets from the original paper on *Progeny Clustering* (Hu et al., 2015) were included in the **progenyClust** package for testing and sharing purposes. As a proof-of-concept example, `test` was a simulated dataset to help users quickly test the algorithm and see how it works. The dataset was generated by randomly drawing 50 samples from bivariate normal distributions with a common identity covariance matrix and a mean at (-1,2), (2,0) and (-1,-2) respectively. Thus, `test` is a 150 by 2 matrix that contains three clusters.

The dataset `cell`, generated experimentally from Slater et al. (2015), contains 444 cell samples and the first three principal components of their morphology metrics. Since the cells were engineered into 4 distinct morphological phenotypes, this dataset in theory should contain 4 clusters. More experimental details of this dataset can be found in Slater et al. (2015) and Hu et al. (2015).

Examples

In this section, we demonstrate the use of the **progenyClust** package in two examples. The first example is a proof-of-concept of how **progenyClust** works on a simulated test dataset. The second example demonstrates the biomedical application of **progenyClust** to identify the number of cell phenotypes based on cell imaging data.

Proof-of-concept example

To show how the `progenyClust()` function works, we use the dataset `test` included in the **progenyClust** package as the input dataset. The goal here is to find the inherent number of clusters present in this dataset, which is known to be three. Since most of the parameters have default values, we can run the `progenyClust()` function for this dataset with the default setting. The R code is as follows:

```
require('progenyClust')
data(test)
set.seed(1)

## run Progeny Clustering with default parameter setting
test.progenyClust <- progenyClust(test)

## plot stability scores computed by Progeny Clustering
plot(test.progenyClust)

## plot clustering results at the optimal cluster number (default)
plot(test.progenyClust, test)

## report the optimal cluster number
summary(test.progenyClust)

## output from the summary
Call:
progenyClust(data = test)

Optimal Number of Clusters:
gap criterion - 3
```

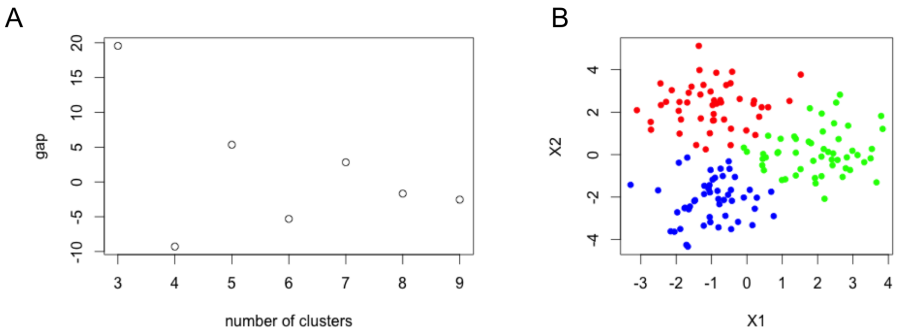


Figure 3: Plots of the "progenyClust" object from clustering the `test` dataset under the default setting. (A) Normalized stability scores based on the 'greatest gap' method were shown at each cluster number. The greater the stability score is, the closer the cluster number matches the true cluster number. (B) The clustered `test` data is shown with the optimal number of clusters.

The summary of the "progenyClust" object concludes that the optimal number for clustering this `test` dataset is three, which agrees with the fact that the dataset was generated from three centers. The plot result of the "progenyClust" object alone is shown in Figure 3A, displaying a curve of normalized stability scores for all candidate numbers of clusters except for the minimum and maximum. This score curve can provide us with insights of clustering quality at all cluster numbers, and help us identify the second preferred number of clusters if needed. Using the `test` data as input, the `plot()`

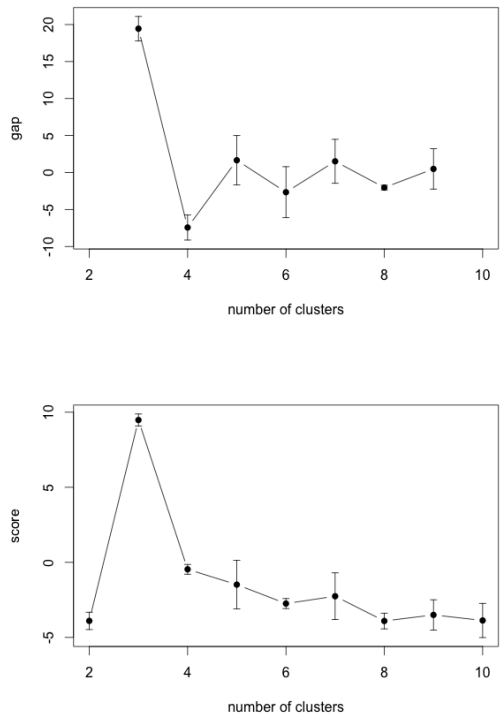


Figure 4: Plot of the "progenyClust" object from running `progenyClust()` on the `test` dataset three times with both evaluation methods, 'greatest gap' (top) and 'greatest score' (bottom). The score curves from both methods estimated that the number of three clusters is best for this dataset. The plot was customized to display the error bars.

function visualizes the data in a scatter plot with three colors, where each corresponds to a cluster (Figure 3B).

Though the default setting of `progenyClust()` function works well in this example, for the purpose of illustrating the capabilities of the function, we will change the input argument values and tune the algorithms slightly. For example, due to the theoretical shortage of the 'greatest gap' criterion, the user might want to obtain estimation from both the 'greatest gap' and the 'greatest score' methods. Though the 'greatest score' method will slow down the algorithm because of the laborious process of generating reference scores, it can evaluate clustering stabilities at the minimum and maximum potential cluster numbers which are ignored by the 'greatest gap' method. The R code for the altered version is shown below. Here, we also change the input argument `repeats` to repeat the algorithm three times instead of one time to obtain standard deviations of the stability scores.

```
set.seed(1)

## run Progeny Clustering with both methods and repeated three times
test2.progenyClust <- progenyClust(test, method = 'both', repeats = 3)

## plot with error bars and summarize the output progenyClust object
plot(test2.progenyClust, errorbar = TRUE, type = 'b')
summary(test2.progenyClust)

## output from the summary
Call:
progenyClust(data = test, method = "both", repeats = 3)

Optimal Number of Clusters:
gap criterion - 3
score criterion -  3
```

It is clear from both the summary and the score curve plots (Figure 4) that both methods agree on the optimal cluster number being three. Specifically, the S3 `plot` method automatically plots two

score curves if the "progenyClust" object was generated with `method = 'both'`. Using the `errbar()` function from **Hmisc**, the S3 `plot` method is able to display error bars with `errorbar = TRUE`.

Application to identifying cell phenotypes

Clustering is a useful technique for the biomedical community, and it can be widely applied to various data-driven research projects. As a second example, we illustrate here how the **progenyClust** package can be used to identify the number of cell phenotypes based on the morphology metrics derived from cell images. In this experiment, biomedical researches used a special technique called "Image Guided Laser Scanning Lithography (LG-LSL)" (Slater et al., 2011) to pattern cells into four shapes. Images of all patterned cells were taken, and morphology metrics were derived to study cytoskeletal and nuclei features of patterned cells. Finding the cell clusters based on their imaging data is of particular interest in this case, and *Progeny Clustering* can help estimate the optimal number for clustering.

Similar to the first example, applying *Progeny Clustering* to the `cell` dataset using the **progenyClust** package is straightforward. The R code is shown below. Here, we use the built-in function `hclust.progenyClust` as `FUNclust` to run the algorithm with hierarchical clustering instead of the default `kmeans`, and we select the optimal cluster number based on the 'greatest gap' criterion. The `plot` and `summary` methods are used to show the output scores and the estimated optimal cluster number. From the output result (Figure 5A), we can see that clustering the cells into four groups has the highest stability, which matches the four patterned cell shapes included in this dataset. The clustering results are shown in Figure 5B in a table of scatter plots for each pairing of variables. Since the cell patterns were engineered, we are fortunate in this example to have prior knowledge of the true number of clusters and to easily test clustering algorithms. However, in a lot of similar biological experiments (e.g. collected tumor cells), we do not possess the knowledge of the true cluster number. In these cases, **progenyClust** can come in handy to identify the optimal cluster number to divide the cells into, and subsequent analyses are then possible for characterizing each cell cluster and discovering its biological or clinical impact.

```
data(cell)
set.seed(1)

## run Progeny Clustering with hierarchical clustering
cell.progenyClust <- progenyClust(cell, hclust.progenyClust)

## plot stability scores, clustering results at optimal cluster number, and summarize results
plot(cell.progenyClust, type = 'b')
plot(cell.progenyClust, cell)
summary(cell.progenyClust)

## output from the summary
Call:
progenyClust(data = cell, FUNclust = hclust.progenyClust)

Optimal Number of Clusters:
gap criterion - 4
```

Summary

This paper introduces the R package **progenyClust**, which identifies the optimal cluster number for any given dataset based on the *Progeny Clustering* algorithm. Improving on the original algorithm, **progenyClust** provides the option to invert stability scores to instability scores, thus preventing the generation of infinite scores in a perfectly stable clustering solution. A variety of parameters (including the clustering method, the evaluation method and the size of progenies) are offered by the package and can be easily adjusted for *Progeny Clustering*. In addition, the default parameter setting specified by the package allows users to perform the algorithm with little background knowledge and parameter tuning. Thanks to the superior computing efficiency of *Progeny Clustering*, this package is a faster alternative to traditional clustering evaluation methods, and it can benefit R communities in biomedicine and beyond.

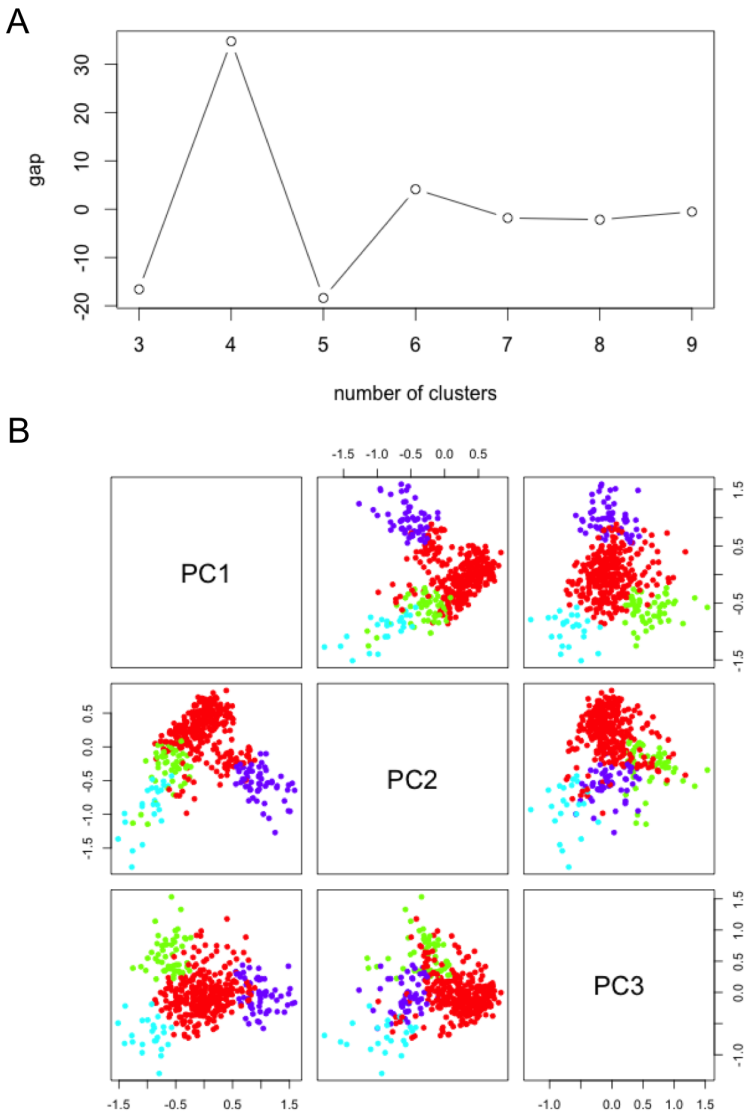


Figure 5: Plots of the "progenyClust" object from running `progenyClust()` on the `cell` dataset with hierarchical clustering. (A) The score curve shows that the `cell` data is best clustered with four clusters. (B) The clustering results with four clusters are shown in a table of scatter plots.

Bibliography

B. Andreopoulos, A. An, X. Wang, and M. Schroeder. A roadmap of clustering algorithms: finding a match for a biomedical application. *Briefings in Bioinformatics*, 10(3):297–314, 2009. [p330]

O. Arbelaitz, I. Gurrutxaga, J. Muguerza, J. M. Pérez, and I. Perona. An extensive comparative study of cluster validity indices. *Pattern Recognition*, 46(1):243–256, 2013. [p328]

A. Ben-Hur, A. Elisseeff, and I. Guyon. A stability based method for discovering structure in clustered data. In *Pacific Symposium on Biocomputing*, volume 7, pages 6–17, 2001. [p328]

E. Budinska, V. Popovici, S. Tejpar, G. D'Ario, N. Lapique, K. O. Sikora, A. F. Di Narzo, P. Yan, J. G. Hodgson, S. Weinrich, et al. Gene expression patterns unveil a new level of molecular heterogeneity in colorectal cancer. *The Journal of Pathology*, 231(1):63–76, 2013. [p328]

M. Charrad, N. Ghazzali, V. Boiteau, and A. Niknafs. Nbclust: an R package for determining the relevant number of clusters in a data set. *Journal of Statistical Software*, 61(6):1–36, 2014a. [p328]

M. Charrad, N. Ghazzali, V. Boiteau, A. Niknafs, and M. M. Charrad. Package 'nbclust'. *J. Stat. Soft*, 61:1–36, 2014b. [p328]

E. Dimitriadou, K. Hornik, and M. K. Hornik. Package 'cclust'. 2015. [p328]

B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315(5814): 972–976, 2007. [p328]

F. E. Harrell Jr and M. F. E. Harrell Jr. Package 'hmisc'. 2015. [p332]

J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979. [p328]

C. Hennig. Package 'fpc'. 2015. [p328]

C. W. Hu, S. M. Kornblau, J. H. Slater, and A. A. Qutub. Progeny clustering: A method to identify biological phenotypes. *Scientific reports*, 5, 2015. [p328, 329, 330, 332]

S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967. [p328]

M. Maechler, P. Rousseeuw, A. Struyf, M. Hubert, K. Hornik, M. Studer, and P. Roudier. Package 'cluster', 2015. [p328]

N. Meinshausen and P. Bühlmann. Stability selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(4):417–473, 2010. [p328]

S. Monti, P. Tamayo, J. Mesirov, and T. Golub. Consensus clustering: a resampling-based method for class discovery and visualization of gene expression microarray data. *Machine Learning*, 52(1-2): 91–118, 2003. [p328]

S. D. Ross. Segmenting sport fans using brand associations: A cluster analysis. *Sport Marketing Quarterly*, 16(1):15, 2007. [p328]

P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. [p328]

J. Slater, J. C. Culver, B. L. Long, C. W. Hu, J. Hu, T. F. Birk, A. A. Qutub, M. E. Dickinson, and J. L. West. Recapitulation and modulation of the cellular architecture of a user-chosen cell-of-interest using cell-derived, biomimetic patterning. *ACS nano*, 2015. [p332]

J. H. Slater, J. S. Miller, S. S. Yu, and J. L. West. Fabrication of multifaceted micropatterned surfaces with laser scanning lithography. *Advanced Functional Materials*, 21(15):2876–2888, 2011. [p335]

T. Sørlie, C. M. Perou, R. Tibshirani, T. Aas, S. Geisler, H. Johnsen, T. Hastie, M. B. Eisen, M. van de Rijn, S. S. Jeffrey, et al. Gene expression patterns of breast carcinomas distinguish tumor subclasses with clinical implications. *Proceedings of the National Academy of Sciences*, 98(19):10869–10874, 2001. [p328]

R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001. [p328, 330]

M. Walesiak, A. Dudek, and M. A. Dudek. Package 'clustersim', 2015. [p328]

Chenyue W. Hu
Rice University
Suite 610, BioScience Research Collaborative, 6500 Main St, Houston, TX 77030
U.S.A
wendyhu001@gmail.com

Amina A. Qutub
Rice University
Suite 610, BioScience Research Collaborative, 6500 Main St, Houston, TX 77030
U.S.A
aminaq@gmail.com

statmod: Probability Calculations for the Inverse Gaussian Distribution

by Göknur Giner and Gordon K. Smyth

Abstract The inverse Gaussian distribution (IGD) is a well known and often used probability distribution for which fully reliable numerical algorithms have not been available. We develop fast, reliable basic probability functions (`dinvgauss`, `pinvgauss`, `qinvgauss` and `rinvgauss`) for the IGD that work for all possible parameter values and which achieve close to full machine accuracy. The most challenging task is to compute quantiles for given cumulative probabilities and we develop a simple but elegant mathematical solution to this problem. We show that Newton's method for finding the quantiles of a IGD always converges monotonically when started from the mode of the distribution. Simple Taylor series expansions are used to improve accuracy on the log-scale. The IGD probability functions provide the same options and obey the same conventions as do probability functions provided in the **stats** package.

Introduction

The inverse Gaussian distribution (IGD) (Tweedie, 1957; Johnson and Kotz, 1970) is widely used in a variety of application areas including reliability and survival analysis (Whitmore, 1975; Chhikara and Folks, 1977; Bardsley, 1980; Chhikara, 1989; Wang and Xu, 2010; Balakrishna and Rahul, 2014). It is more generally used for modeling non-negative positively skewed data because of its connections to exponential families and generalized linear models (Seshadri, 1993; Blough et al., 1999; Smyth and Verbyla, 1999; De Jong and Heller, 2008).

Basic probability functions for the IGD have been implemented previously in James Lindsey's R package **rmutil** (Lindsey, 2010) and in the CRAN packages **SuppDists** (Wheeler, 2009) and **STAR** Pouzat (2012). We have found however that none of these IGD functions work for all parameter values or return results to full machine accuracy. Bob Wheeler remarks in the **SuppDists** documentation that the IGD "is an extremely difficult distribution to treat numerically". The **rmutil** package was removed from CRAN in 1999 but is still available from Lindsey's webpage. **SuppDists** was orphaned in 2013 but is still available from CRAN. The **SuppDists** code is mostly implemented in C while the other packages are pure R as far as the IGD functions are concerned.

The probability density of the IGD has a simple closed form expression and so is easy to compute. Care is still required though to handle infinite parameter values that correspond to valid limiting cases. The cumulative distribution function (cdf) is also available in closed form via an indirect relationship with the normal distribution (Shuster, 1968; Chhikara and Folks, 1974). Considerable care is nevertheless required to compute probabilities accurately on the log-scale, because the formula involves a sum of two normal probabilities on the un-logged scale. Random variates from IGDs can be generated using a combination of chisquare and binomial random variables (Michael et al., 1976). Most difficult is the inverse cdf or quantile function, which must be computed by some iterative numerical approximation.

Two strategies have been used to compute IGD quantiles. One is to solve for the quantile using a general-purpose equation solver such as the `uniroot` function in R. This is the approach taken by the `qinvgauss` functions in the **rmutil** and **STAR** packages. This approach can usually be relied on to converge satisfactorily but is computationally slow and provides only limited precision. The other approach is to use Newton's method to solve the equation after applying an initial approximation (Kallioras and Koutrouvelis, 2014). This approach was taken by one of the current authors when developing inverse Gaussian code for S-PLUS (Smyth, 1998). It is also the approach taken by the `qinvGauss` function in the **SuppDists** package. This approach is fast and accurate when it works but can fail unpredictably when the Newton iteration diverges. Newton's method cannot in general be guaranteed to converge, even when the initial approximation is close to the required value, and the parameter values for which divergence occurs are hard to predict.

We have resolved the above difficulties by developing a Newton iteration for the IGD quantiles that has guaranteed convergence. Instead of attempting to find a starting value that is close to the required solution, we instead use the convexity properties of the cdf function to approach the required quantiles in a predictable fashion. We show that Newton's method for finding the quantiles of an IGD always converges when started from the mode of the distribution. Furthermore the convergence is monotonic, so that backtracking is eliminated. Newton's method is eventually quadratically convergent, meaning that the number of decimal places corrected determined tends to double with each iteration (Press et al., 1992). Although the starting value may be far from the required solution, the rapid convergence

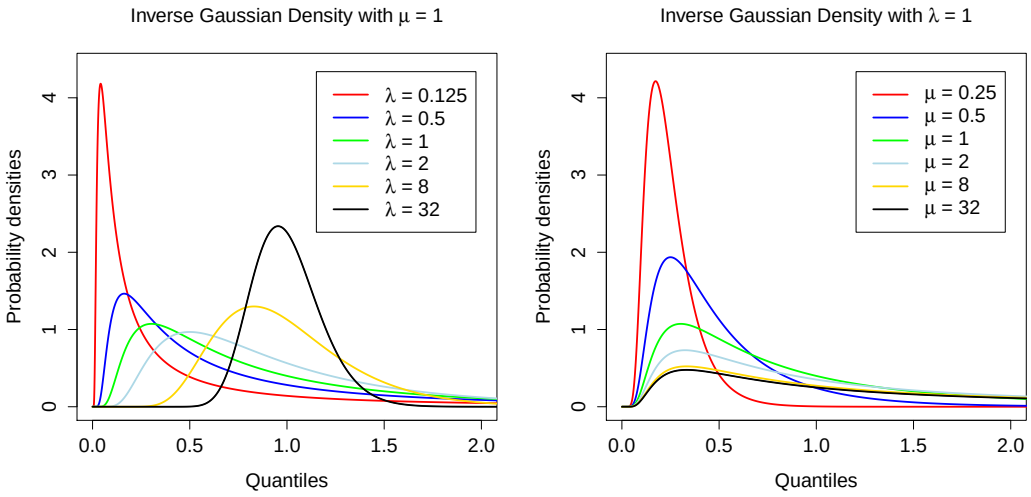


Figure 1: Probability density functions of inverse Gaussian distributions. The left panel shows densities for different λ with $\mu = 1$. The right panel shows densities for different μ for $\lambda = 1$. The densities are unimodal with mode between 0 and μ. As μ/λ increases the distribution becomes more right skew and the mode decreases relative to the mean. Note that $\lambda = 1/\phi$.

means the starting value is quickly left behind. Convergence tends to be rapid even when the required quantile in the extreme tails of the distribution.

The above methods have been implemented in the `dinvgauss`, `pinvgauss`, `qinvgauss` and `rinvgauss` functions of the **statmod** package (Smyth, 2016). The functions give close to machine accuracy for all possible parameter values. They obey similar conventions to the probability functions provided in the **stats** package. Tests show that the functions are faster, more accurate and more reliable than existing functions for the IGD. Every effort has to made to ensure that the functions return results for the widest possible range of parameter values.

Density function

The inverse Gaussian distribution, denoted IG(μ,ϕ), has probability density function (pdf)

$$d(x;\mu,\phi) = \left(2\pi\phi x^3\right)^{-1/2} \exp\left\{-\frac{(x-\mu)^2}{2\phi\mu^2 x}\right\} \tag{1}$$

for $x > 0$, $\mu > 0$ and $\phi > 0$. The mean of the distribution is μ and the variance is $\phi\mu^3$. In generalized linear model theory (McCullagh and Nelder, 1989; Smyth and Verbyla, 1999), ϕ is called the *dispersion* parameter. Another popular parametrization of the IGD uses $\lambda = 1/\phi$, which we call the *shape* parameter. For best accuracy, we compute $d(x;\mu,\phi)$ on the log-scale and then exponentiate if an unlogged value is required.

Note that the mean μ can be viewed as a scaling parameter: if X is distributed as IG(μ,ϕ), then X/μ is also inverse Gaussian with mean 1 and dispersion $\phi\mu$. The skewness of the distribution is therefore determined by $\phi\mu$, and in fact $\phi\mu$ is the squared coefficient of variation of the distribution.

The IGD is unimodal with mode at

$$m = \mu\left\{\left(1+\kappa^2\right)^{1/2} - \kappa\right\} \tag{2}$$

where $\kappa = 3\phi\mu/2$ (Johnson and Kotz, 1970, p. 142). The second factor in the mode is strictly between 0 and 1, showing that the mode is strictly between 0 and μ. Figure 1 shows the pdf of the IGD for various choices of μ and λ.

Care needs to be taken with special cases when evaluating the pdf (Table 1). When $\phi\mu$ is large, a Taylor series expansion shows that the mode becomes dependent on ϕ only:

$$m = \mu\kappa\left\{\left(1+\kappa^{-2}\right)^{1/2} - 1\right\} = \mu\kappa\left(\frac{1}{2\kappa^2} - \frac{1}{8\kappa^4} + \frac{1}{16\kappa^6} - \cdots\right) \approx \mu\kappa\frac{1}{2\kappa^2} = \frac{1}{3\phi}. \tag{3}$$

Description	Parameter values	log-pdf	pdf	cdf
Left limit	$x < 0$	$-\infty$	0	0
Left limit	$x = 0$, $\mu > 0$ and $\phi < \infty$	$-\infty$	0	0
Left limit	$x < \mu$ and $\phi = 0$	$-\infty$	0	0
Right limit	$x = \infty$	$-\infty$	0	1
Right limit	$x > \mu$ and $\phi = 0$	$-\infty$	0	1
Right limit	$x > 0$ and $\phi = \infty$	$-\infty$	0	1
Spike	$x = \mu < \infty$ and $\phi = 0$	∞	∞	1
Spike	$x = 0$ and $\phi = \infty$	∞	∞	1
Inverse chisquare	$\mu = \infty$ and $\phi < \infty$	Eqn 5	Eqn 5	Uses `pchisq`
Invalid	$\mu < 0$ or $\phi < 0$	`NA`	`NA`	`NA`

Table 1: Probability density function values for special cases of the parameter values. The pdf values for infinite parameters are theoretical limit values.

Under the same conditions, the peak value of the density can be seen to converge to $\phi(2\pi/27)^{-1/2} \times \exp(-3/2)$. This shows that the distribution has a spike at 0 whenever ϕ is very large, regardless of μ. It is also known that

$$\frac{(X-\mu)^2}{\phi X \mu^2} \sim \chi^2_1 \tag{4}$$

(Shuster, 1968). Amongst other things, this implies that $1/(X\phi) \sim \chi^2_1$ asymptotically for μ large. For infinite μ, the density becomes

$$d(x;\infty,\phi) = \left(2\pi x^3 \phi\right)^{-1/2} \exp\left(-\frac{1}{2\phi x}\right). \tag{5}$$

The pdf is always `NA` if x is `NA`. Missing values for ϕ lead to `NA` values for the pdf except when $x < 0$ or $x = \infty$. Missing values for μ lead to `NA` values for the pdf except when $x < 0$, $x = \infty$ or $\phi = \infty$.

Next we give some code examples. We start by loading the packages that we will compare. Note that **statmod** is loaded last and is therefore first in the search path.

```
> library(rmutil)
> library(SuppDists)
> library(STAR)
> library(statmod)
```

The **statmod** `dinvgauss` function checks for out-of-range or missing values:

```
> options(digits = 3)
> dinvgauss(c(-1, 0, 1, 2, Inf, NA), mean = 1.5, dispersion = 0.7)
[1] 0.000 0.000 0.440 0.162 0.000    NA
```

Infinite mean corresponds to an inverse-chisquare case:

```
> dinvgauss(c(-1, 0, 1, 2, Inf, NA), mean = Inf, dispersion = 0.7)
[1] 0.000 0.000 0.233 0.118 0.000    NA
```

Infinite dispersion corresponds to a spike at 0 regardless of the mean:

```
> dinvgauss(c(-1, 0, 1, 2, Inf, NA), mean = NA, dispersion = Inf)
[1]   0 Inf   0   0   0  NA
```

Extreme x values have zero density regardless of the mean or dispersion:

```
> dinvgauss(c(-1, 0, 1, Inf), mean = NA, dispersion = NA)
[1]  0 NA NA  0
```

All the existing functions `rmutil::dinvgauss`, `SuppDist::dinvGauss` and `STAR::dinvgauss` return errors for the above calls; they do not tolerate `NA` values, or infinite parameter values, or x values outside the support of the distribution.

Cumulative distribution function

Let $p(q;\mu,\phi) = P(X \le q)$ be the left tail cdf, and write $\bar{p}(q;\mu,\phi)$ for the right tail probability $P(X > q) = 1 - p(q;\mu,\phi)$. The formula developed by Shuster (1968) for the cdf is

$$p(q;\mu,\phi) = p_{\text{norm}}((q_m - 1)/r) + \exp(2/\phi_m)p_{\text{norm}}(-(q_m + 1)/r)$$

where $q_m = q/\mu$, $\phi_m = \phi\mu$, $r = (q\phi)^{1/2}$ and p_{norm} is the cdf of the standard normal distribution. The right tail probability can be written similarly:

$$\bar{p}(q;\mu,\phi) = \bar{p}_{\text{norm}}((q_m - 1)/r) - \exp(2/\phi_m)p_{\text{norm}}(-(q_m + 1)/r)$$

where $\bar{p}_{\text{norm}}$ is the right tail of the standard normal. The fact that this formula is additive on the unlogged scale poses some numerical problems. The $p_{\text{norm}}()$ evaluations are subject to floating underflow, the $\exp()$ evaluation is subject to overflow, and there is the danger of subtractive cancellation when computing the right tail probability.

It is possible to derive an asymptotic expression for the right tail probability. If q is very large then:

$$\log \bar{p}(q;1,\phi) \approx \frac{1}{\phi_m} - 0.5\log\pi - \log(2\phi_m) - 1.5\log\left(\frac{q_m}{2\phi_m} + 1\right) - \frac{q_m}{2\phi_m}.$$

See the Appendix for the derivation of this approximation. This approximation is very accurate when $\phi_m^{-1/2}(q_m - 1) > 10^5$, but only gives 2–3 significant figures correctly for more modest values such as $\phi_m^{-1/2}(q_m - 1) = 10$.

To avoid or minimize the numerical problems described above, we convert the terms in the cdf to the log-scale and remove a common factor before combining the two term terms to get $\log p$. Given a quantile value q, we compute the corresponding $\log p$ as follows:

$$\begin{aligned} a &= \log p_{\text{norm}}((q_m - 1)/r) \\ b &= 2/\phi_m + \log p_{\text{norm}}(-(q_m + 1)/r) \\ \log p &= a + \text{log1p}(\exp(b - a)) \end{aligned}$$

where $\log p_{\text{norm}}()$ is computed by `pnorm` with `lower.tail=TRUE` and `log.p=TRUE`. Note also that `log1p()` is an R function that computes the logarithm of one plus its argument avoiding subtractive cancellation for small arguments. The computation of the right tail probability is similar but with

$$\begin{aligned} a &= \log \bar{p}_{\text{norm}}((q_m - 1)/r) \\ \log \bar{p} &= a + \text{log1p}(-\exp(b - a)). \end{aligned}$$

Because of this careful computation, `statmod::pinvgauss` function is able to compute correct cdf values even in the far tails of the distribution:

```
> options(digits = 4)
> pinvgauss(0.001, mean = 1.5, disp = 0.7)
[1] 3.368e-312
> pinvgauss(110, mean = 1.5, disp = 0.7, lower.tail = FALSE)
[1] 2.197e-18
```

None of the existing functions can distinguish such small left tail probabilities from zero:

```
> rmutil::pinvgauss(0.001, m = 1.5, s = 0.7)
[1] 0
> SuppDists::pinvGauss(0.001, nu = 1.5, lambda = 1/0.7)
[1] 0
> STAR::pinvgauss(0.001, mu = 1.5, sigma2 = 0.7)
[1] 0
```

`rmutil::pinvgauss` doesn't compute right tail probabilities. `STAR::pinvgauss` does but can't distinguish right tail probabilities less than `1e-17` from zero:

```
> STAR::pinvgauss(110, mu = 1.5, sigma2 = 0.7, lower.tail = FALSE)
[1] 0
```

`SuppDists::pinvGauss` returns non-zero right tail probabilities, but these are too large by a factor of 10:

```
> SuppDists::pinvGauss(110, nu = 1.5, lambda = 1/0.7, lower.tail = FALSE)
[1] 2.935e-17
```

The use of log-scale computations means that `statmod::pinvgauss` can accurately compute log-probabilities that are too small to be represented on the unlogged scale:

```
> pinvgauss(0.0001, mean = 1.5, disp = 0.7, log.p = TRUE)
[1] -7146.914
```

None of the other packages can compute log-probabilities less than about -700.

`pinvgauss` handles special cases similarly to `dinvgauss` (Table 1). Again, none of the existing functions do this:

```
> pinvgauss(c(-1, 0, 1, 2, Inf, NA), mean = 1.5, dispersion = 0.7)
[1] 0.0000 0.0000 0.5009 0.7742 1.0000     NA
```

Infinite mean corresponds to an inverse-chisquare case:

```
> pinvgauss(c(-1, 0, 1, 2, Inf, NA), mean = Inf, dispersion = 0.7)
[1] 0.000 0.000 0.232 0.398 1.000    NA
```

Infinite dispersion corresponds to a spike at 0 regardless of the mean:

```
> pinvgauss(c(-1, 0, 1, 2, Inf, NA), mean = NA, dispersion = Inf)
[1]  0  1  1  1  1 NA
```

Extreme x values have cdf equal to 0 or 1 regardless of the mean or dispersion:

```
> pinvgauss(c(-1, 0, 1, Inf), mean = NA, dispersion = NA)
[1]  0 NA NA  1
```

We can test the accuracy of the cdf functions by comparing to the cdf of the χ^2_1 distribution. For any $q_1 < \mu$, let $q_2 > \mu$ be that value satisfying

$$z = \frac{(q_1 - \mu)^2}{\phi\mu^2 q_1} = \frac{(q_2 - \mu)^2}{\phi\mu^2 q_2}.$$

From equation 4, we can conclude that the upper tail probability for the χ^2_1 distribution at z should be the sum of the IGD tail probabilities for q_1 and q_2, i.e.,

$$\bar{p}_{chisq}(z) = p(q_1; \mu, \phi) + \bar{p}(q_2; \mu, \phi). \tag{6}$$

The following code implements this process for an illustrative example with $\mu = 1.5$, $\phi = 0.7$ and $q_1 = 0.1$. First we have to solve for q_2:

```
> options(digits = 4)
> mu <- 1.5
> phi <- 0.7
> q1 <- 0.1
> z <- (q1 - mu)^2 / (phi * mu^2 * q1)
> polycoef <- c(mu^2, -2 * mu - phi * mu^2 * z, 1)
> q <- Re(polyroot(polycoef))
> q
[1]  0.1 22.5
```

The chisquare cdf value corresponding to the left hand size of equation 6 is:

```
> options(digits = 18)
> pchisq(z, df = 1, lower.tail = FALSE)
[1] 0.00041923696954098788
```

Now we compute the right hand size of equation 6 using each of the IGD packages, starting with **statmod**:

```
> pinvgauss(q[1], mean = mu, disp = phi) +
+ pinvgauss(q[2], mean = mu, disp = phi, lower.tail = FALSE)
[1] 0.00041923696954098701
> rmutil::pinvgauss(q[1], m = mu, s = phi) +
+ 1 - rmutil::pinvgauss(q[2], m = mu, s = phi)
[1] 0.00041923696954104805
> SuppDists::pinvGauss(q[1], nu = mu, lambda = 1/phi) +
```

```
+ SuppDists::pinvGauss(q[2], nu = mu, lambda = 1/phi, lower.tail = FALSE)
[1] 0.00041923696954101699
> STAR::pinvgauss(q[1], mu = mu, sigma2 = phi) +
+ STAR::pinvgauss(q[2], mu = mu, sigma2 = phi, lower.tail = FALSE)
[1] 0.00041923696954100208
```

It can be seen that the **statmod** function is the only one to agree with pchisq to 15 significant figures, corresponding to a relative error of about 10^{-15}. The other three packages give 12 significant figures, corresponding to relative errors of slightly over 10^{-12}.

More extreme tail values give even more striking results. We repeat the above process now with $q_1 = 0.01$:

```
> q1 <- 0.01
> z <- (q1 - mu)^2 / (phi * mu^2 * q1)
> polycoef <- c(mu^2, -2 * mu - phi * mu^2 * z, 1)
> q <- Re(polyroot(polycoef))
```

The reference chisquare cdf value is:

```
> pchisq(z, df = 1, lower.tail = FALSE)
[1] 1.6427313604456241e-32
```

This can be compared to the corresponding values from the IGD packages:

```
> pinvgauss(q[1], mean = mu, disp = phi) +
+ pinvgauss(q[2], mean = mu, disp = phi, lower.tail = FALSE)
[1] 1.6427313604456183e-32
> rmutil::pinvgauss(q[1], m = mu, s = phi) +
+ 1 - rmutil::pinvgauss(q[2], m = mu, s = phi)
[1] 0
> SuppDists::pinvGauss(q[1], nu = mu, lambda = 1/phi) +
+ SuppDists::pinvGauss(q[2], nu = mu, lambda = 1/phi, lower.tail = FALSE)
[1] 8.2136568022278466e-33
> STAR::pinvgauss(q[1], mu = mu, sigma2 = phi) +
+ STAR::pinvgauss(q[2], mu = mu, sigma2 = phi, lower.tail = FALSE)
[1] 1.6319986233795599e-32
```

It can be seen from the above that **rmutil** and **SuppDists** do not agree with pchisq to any significant figures, meaning that the relative error is close to 100%, while **STAR** manages 3 significant figures. **statmod** on the other hand continues to agree with pchisq to 15 significant figures.

Inverting the cdf

Now consider the problem of computing the quantile function $q(p;\mu,\phi)$. The quantile function computes q satisfying $P(X \le q) = p$.

If q_n is an initial approximation to q, then Newton's method is a natural choice for refining the estimate. Newton's method gives the updated estimate as

$$q_{n+1} = q_n + \frac{p - p(q_n;\mu,\phi)}{d(q_n;\mu,\phi)}.$$

For right-tail probabilities, the Newton step is almost the same:

$$q_{n+1} = q_n - \frac{p - \bar{p}(q_n;\mu,\phi)}{d(q_n;\mu,\phi)}$$

where now $P(X > q) = p$. Newton's method is very attractive because it is quadratically convergent if started sufficiently close to the required value. It is hard however to characterize how close the starting value needs to be to achieve convergence and in general there is no guarantee that the Newton iteration will not diverge or give impossible values such as $q < 0$ or $q = \infty$. Our approach is to derive simple conditions on the starting values such that the Newton iteration always converges and does so without any backtracking. We call this behavior *monotonic convergence*.

Recall that the IGD is unimodal for all parameter values with mode m given previously. It follows that the pdf $d(q;\mu\phi)$ is increasing for all $q < m$ and decreasing for all $q > m$ and the cdf $p(q;\mu,\phi)$ is convex for $q < m$ and concave for $q > m$. In other words, the cdf has a point of inflexion at the mode of the distribution.

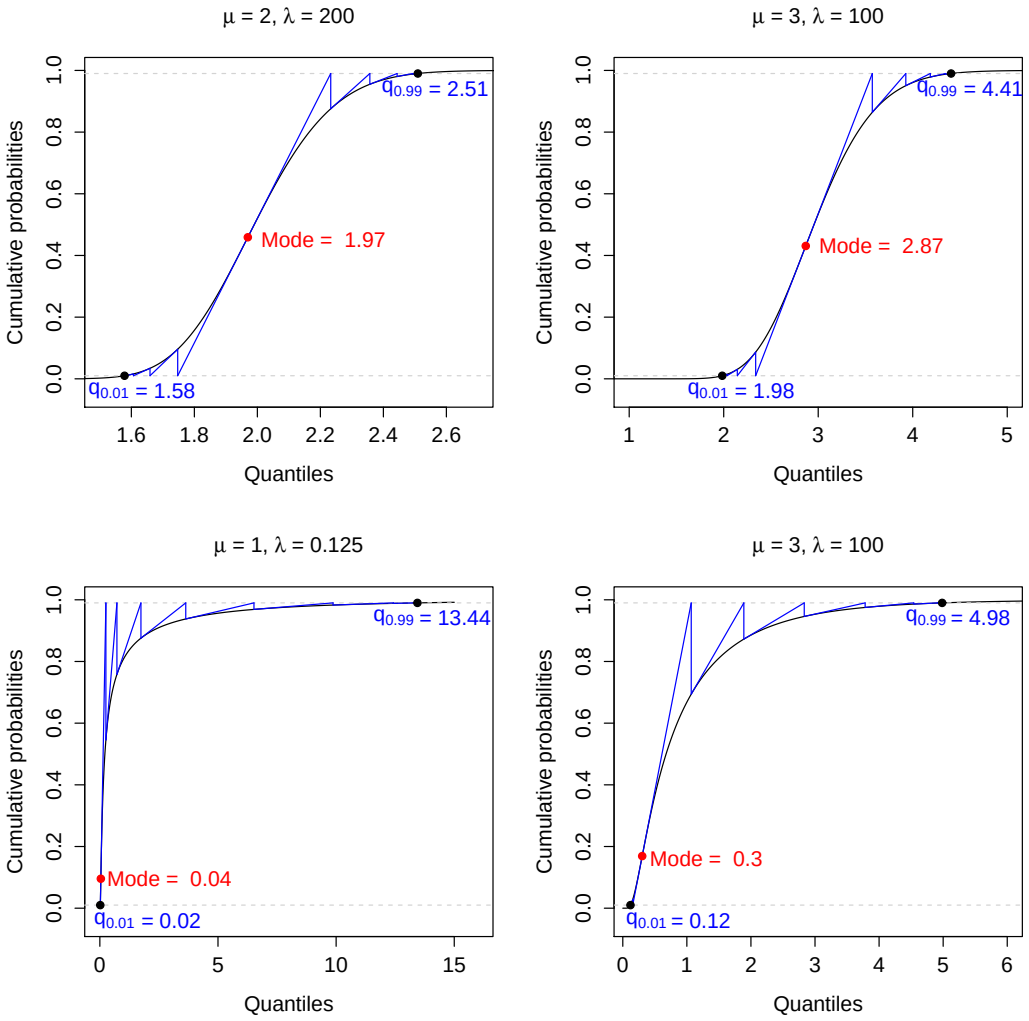


Figure 2: Monotonic Newton's method for quantiles of inverse Gaussian distributions. The cdf has a point of inflexion, marked by a red dot, at the mode of the distribution. Blue lines show the progress of the iteration for the 0.01 or 0.99 quantiles. Since the cdf is convex to the left of the mode and concave to the right, starting the iteration at the point of inflexion ensures convergence to the required quantiles without any backtracking.

Suppose that the required q satisfies $q \geq m$ and suppose that the working estimate satisfies $m \leq q_n \leq q$. It can be seen that the cdf is concave in the interval $[q_n, q]$, the Newton step will be positive and the updated estimate q_{n+1} will still satisfy $m \leq q_{n+1} \leq q$ (Figure 2). Suppose instead that $q < m$ and suppose that the working estimate satisfies $q \leq q_n \leq m$. In this case it can be seen that the cdf is convex in the interval $[q_n, q]$, the Newton step will be negative and the updated estimate q_n will still satisfy $q \leq q_{n+1} \leq m$ (Figure 2). It follows that Newton's method is always monotonically convergent provided that the starting value lies between the mode m and the required value q. In fact the mode m itself can be used as the starting value. Note that to compute the mode m accurately without subtractive cancellation we use equation 3 when κ is large and use equation 2 otherwise.

We use $q_0 = m$ as the starting value for the Newton iteration unless the left or right tail probability is very small. When the left tail probability is less than 10^{-5}, we use instead

$$q_0 = \frac{\mu}{\phi q_{\text{norm}}^2}$$

where q_{norm} is the corresponding quantile of the standard normal distribution. When the right tail probability is less than 10^{-5}, we use

$$q_0 = q_{\text{gamma}}$$

where q_{gamma} is the corresponding quantile of the gamma distribution with the same mean and variances as the IGD. These starting values are closer to the required q than is m but still lie between m and the required q and so are in the domain of monotonic convergence. We use the alterative starting values only for extreme tail probabilities because in other cases the computational cost of computing the starting value is greater than the saving enjoyed by reducing the number of Newton iterations that are needed.

The term $p - p(q_n; \mu, \phi)$ in the Newton step could potentially suffer loss of floating point precision by subtractive cancellation when p and $p(q_n; \mu, \phi)$ are nearly equal or if p is very close to 1. To avoid this we work with p on the log-scale and employ a Taylor series expansion when p and $p(q_n; \mu, \phi)$ are relatively close. Let $\delta = \log p - \log p(q_n; \mu, \phi)$. When $|\delta| < 10^{-5}$, we approximate

$$p - p(q_n; \mu, \phi) \approx \delta \exp\left\{\log p + \text{log1p}(-\delta/2)\right\}.$$

Here $\log p(q_n; \mu, \phi)$ is computed by `pinvgauss` with `log.p=TRUE` and $\text{log1p}(-\delta/2)$ is computed using the `log1p` function.

We find that the **statmod** `qinvgauss` package gives 16 significant figures whereas the other packages give no more than 6–8 figures of accuracy. Precision can be demonstrated by comparing the probability vector p with the values obtained by passing the probabilities through `qinvgauss` and `pinvgauss`. `qinvgauss` and `pinvgauss` are inverse functions, so the final probabilities should be equal in principle to the original values. Error is measured by comparing the original and processed probability vectors:

```
> p <- c(0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 0.5,
+        0.9, 0.99, 0.999, 0.9999, 0.99999, 0.999999)
>
> p1 <- pinvgauss(qinvgauss(p, mean = 1, disp = 1), mean = 1, disp = 1)
> p2 <- rmutil::pinvgauss(rmutil::qinvgauss(p, m = 1, s = 1), m = 1, s = 1)
> p3 <- SuppDists::pinvGauss(SuppDists::qinvGauss(p, nu = 1, la = 1), nu = 1, la = 1)
> p4 <- STAR::pinvgauss(STAR::qinvgauss(p, mu = 1, sigma2 = 1), mu = 1, sigma2 = 1)
>
> options(digits = 4)
> summary( abs(p-p1) )
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.00e+00 0.00e+00 0.00e+00 1.92e-17 2.20e-19 2.22e-16
> summary( abs(p-p2) )
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.00e+00 5.10e-09 8.39e-08 3.28e-07 5.92e-07 1.18e-06
> summary( abs(p-p3) )
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
1.00e-12 6.00e-12 2.77e-10 1.77e-09 2.58e-09 1.03e-08
> summary( abs(p-p4) )
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.00e+00 0.00e+00 1.20e-08 8.95e-07 2.17e-07 6.65e-06
```

It can be seen that the error for `statmod::qinvgauss` is never greater than `2e-16`.

Similar results are observed if relative error is assessed in terms of the quantile q instead of the probability p:

```
> q <- qinvgauss(p, mean = 1, disp = 1)
> q1 <- qinvgauss(pinvgauss(q, mean = 1, disp = 1), mean = 1, disp = 1)
> q2 <- rmutil::qinvgauss(rmutil::pinvgauss(q, m = 1, s = 1), m = 1, s = 1)
> q3 <- SuppDists::qinvGauss(SuppDists::pinvGauss(q, nu = 1, la = 1), nu = 1, la = 1)
> q4 <- STAR::qinvgauss(STAR::pinvgauss(q, mu = 1, sigma2 = 1), mu = 1, sigma2 = 1)
> summary( abs(q1-q)/q )
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.00e+00 0.00e+00 0.00e+00 5.57e-17 0.00e+00 4.93e-16
> summary( abs(q2-q)/q )
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.00e+00 1.70e-06 3.30e-06 8.94e-05 8.80e-05 5.98e-04
> summary( abs(q3-q)/q )
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
1.09e-08 3.94e-08 4.78e-08 4.67e-08 5.67e-08 8.93e-08
> summary( abs(q4-q)/q )
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.00e+00 3.00e-07 1.40e-06 9.20e-05 9.42e-05 5.46e-04
```

The relative error for `statmod::qinvgauss` is never worse than `5e-16`.

Speed was determined by generating p as a vector of a million random uniform deviates, and running the `qinvgauss` or `qinvGauss` functions on p with mean and dispersion both equal to one.

```
> set.seed(20140526)
> u <- runif(1000)
> p <- runif(1e6)
> system.time(q1 <- qinvgauss(p, mean = 1, shape = 1))
   user  system elapsed
   4.29    0.41    4.69
> system.time(q2 <- rmutil::qinvgauss(p, m = 1, s = 1))
   user  system elapsed
 157.39    0.03  157.90
> system.time(q3 <- SuppDists::qinvGauss(p, nu = 1, lambda = 1))
   user  system elapsed
  13.59    0.00   13.68
> system.time(q4 <- STAR::qinvgauss(p, mu = 1, sigma2 = 1))
   user  system elapsed
 266.41    0.06  267.25
```

Timings shown here are for a Windows laptop with a 2.7GHz Intel i7 processor running 64-bit R-devel (built 31 January 2016). The **statmod** qinvgauss function is 40 times faster than the **rmutil** or **STAR** functions about 3 times faster than **SuppDists**.

Reliability is perhaps even more crucial than precision or speed. `SuppDists::qinvGauss` fails for some parameter values because Newton's method does not converge from the starting values provided:

```
> options(digits = 4)
> SuppDists::qinvGauss(0.00013, nu=1, lambda=3)
Error in SuppDists::qinvGauss(0.00013, nu = 1, lambda = 3) :
Iteration limit exceeded in NewtonRoot()
```

By contrast, `statmod::qinvgauss` runs successfully for all parameter values because divergence of the algorithm is impossible:

```
> qinvgauss(0.00013, mean = 1, shape = 3)
[1] 0.1504
```

`qinvgauss` returns right tail values accurately, for example:

```
> qinvgauss(1e-20, mean = 1.5, disp = 0.7, lower.tail = FALSE)
[1] 126.3
```

The same probability can be supplied as a left tail probability on the log-scale, with the same result:

```
> qinvgauss(-1e-20, mean = 1.5, disp = 0.7, log.p = TRUE)
[1] 126.3
```

Note that `qinvgauss` returns the correct quantile in this case even though the left tail probability is not distinguishable from 1 in floating point arithmetic on the unlogged scale. By contrast, the **rmutil**

and **STAR** functions do not compute right tail values and the **SuppDists** function fails to converge for small right tail probabilities:

```
> SuppDists::qinvGauss(1e-20, nu = 1.5, lambda = 1/0.7, lower.tail = FALSE)
Error in SuppDists::qinvGauss(1e-20, nu = 1.5, lambda = 1/0.7, lower.tail = FALSE) :
Infinite value in NewtonRoot()
```

Similarly for log-probabilities, the **rmutil** and **STAR** functions do not accept log-probabilities and the **SuppDists** function gives an error:

```
> SuppDists::qinvGauss(-1e-20, nu = 1.5, lambda = 1/0.7, log.p=TRUE)
Error in SuppDists::qinvGauss(-1e-20, nu = 1.5, lambda = 1/0.7, log.p = TRUE) :
Infinite value in NewtonRoot()
```

All the **statmod** IGD functions allow variability to be specified either by way of a dispersion (ϕ) or shape (λ) parameter:

```
> args(qinvgauss)
function (p, mean = 1, shape = NULL, dispersion = 1, lower.tail = TRUE,
    log.p = FALSE, maxit = 200L, tol = 1e-14, trace = FALSE)
```

Boundary or invalid p are detected:

```
> options(digits = 4)
> qinvgauss(c(0, 0.5, 1, 2, NA))
[1] 0.0000 0.6758    Inf     NA     NA
```

as are invalid values for μ or ϕ:

```
> qinvgauss(0.5, mean = c(0, 1, 2))
[1]     NA 0.6758 1.0285
```

The **statmod** functions `dinvgauss`, `pinvgauss` and `qinvgauss` all preserve the attributes of the first input argument provided that none of the other arguments have longer length. For example, `qinvgauss` will return a matrix if p is a matrix:

```
> p <- matrix(c(0.1, 0.6, 0.7, 0.9), 2, 2)
> rownames(p) <- c("A", "B")
> colnames(p) <- c("X1", "X2")
> p
      X1     X2
A 0.6001 0.3435
B 0.4919 0.4987
> qinvgauss(p)
      X1     X2
A 0.8486 0.4759
B 0.6637 0.6739
```

Similarly the names of a vector are preserved on output:

```
> p <- c(0.1, 0.6, 0.7, 0.9)
> names(p) <- LETTERS[1:4]
> qinvgauss(p)
     A      B      C      D
0.2376 0.8483 1.0851 2.1430
```

Random deviates

The functions `statmod::rinvgauss`, `SuppDists::rinvGauss` and `STAR::rinvgauss` all use the same algorithm to compute random deviates from the IGD. The method is to generate chisquare random deviates corresponding to $(X-\mu)^2/(\phi X\mu^2)$, and then choose between the two possible X values leading to the same chisquare value with probabilities worked out by Michael et al. (1976). The **SuppDists** function is faster than the others because of the implementation in C. Nevertheless, the pure R **statmod** and **STAR** functions are acceptably fast. The **statmod** function generates a million random deviates in about a quarter of a second of elapsed time on a standard business laptop computer while **STAR** takes about half a second.

The `rmutil::rinvgauss` function generates random deviates by running `qinvgauss` on random uniform deviates. This is far slower and less accurate than the other functions.

Discussion

Basic probability calculations for the IGD have been available in various forms for some time but the functions described here are the first to work for all parameter values and to return close to full machine accuracy.

The **statmod** functions achieve good accuracy by computing probabilities on the log-scale where possible. Care is given to handle special limiting cases, including some cases that have not been previously described. The **statmod** functions trap invalid parameter values, provide all the standard arguments for probability functions in the R and preserve argument attributes on output.

A new strategy has been described to invert the cdf using a monotonically convergent Newton iteration. It may seem surprising that we recommend starting the iteration from the same value regardless of the quantile required. Intuitively, a starting value that is closer to the required quantile might have been expected to be better. However using an initial approximation runs the risk of divergence, and convergence of Newton's method from the mode is so rapid that the potential advantage of a closer initial approximation is minimized. The **statmod** `qinvgauss` function is 40 times faster than the quantile functions in the **rmutil** or **STAR** packages, despite returning 16 rather than 6 figures of accuracy. It is also 3 times faster than **SuppDists**, even though `SuppDists::qinvGauss` is written in C, uses the same basic Newton strategy and has a less stringent stopping criterion. The starting values for Newton's method used by `SuppDists::qinvGauss` are actually closer to the final values than those used by `statmod::qinvgauss`, but the latter are more carefully chosen to achieve smooth convergence without backtracking. `SuppDists::qinvGauss` uses the log-normal approximation of Whitmore and Yalovsky (1978) to start the Newton iteration and the `STAR::qinvgauss` uses the same approximation to setup the interval limits for `uniroot`. Unfortunately the log-normal approximation has much heavier tails than the IGD, meaning that the starting values are more extreme than the required quantiles and are therefore outside the domain of monotonic convergence.

As well as the efficiency gained by avoiding backtracking, monotonic convergence has the advantage that any change in sign of the Newton step is a symptom that the limits of floating point accuracy have been reached. In the **statmod** `qinvgauss` function, the Newton iteration is stopped if this change of sign occurs before the convergence criterion is achieved.

The current **statmod** functions could be made faster by reimplementing in C, but the pure R versions have benefits in terms of understandability and easy maintenance, and they are only slightly slower than comparable functions such as `qchisq` and `qt`.

This strategy used here to compute the quantile could be used for any continuous unimodal distribution, or for continuous distribution that can be transformed to be unimodal.

```
> sessionInfo()
R Under development (unstable) (2016-01-31 r70055)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 7 x64 (build 7601) Service Pack 1

locale:
[1] LC_COLLATE=English_Australia.1252  LC_CTYPE=English_Australia.1252
[3] LC_MONETARY=English_Australia.1252 LC_NUMERIC=C
[5] LC_TIME=English_Australia.1252

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] statmod_1.4.24    STAR_0.3-7        codetools_0.2-14  gss_2.1-5
 [5] R2HTML_2.3.1      mgcv_1.8-11       nlme_3.1-124      survival_2.38-3
 [9] SuppDists_1.1-9.2 rmutil_1.0

loaded via a namespace (and not attached):
[1] Matrix_1.2-3    splines_3.3.0   grid_3.3.0      lattice_0.20-33
```

Bibliography

N. Balakrishna and T. Rahul. Inverse Gaussian distribution for modeling conditional durations in finance. *Communications in Statistics-Simulation and Computation*, 43(3):476–486, 2014. [p339]

W. Bardsley. Note on the use of the inverse Gaussian distribution for wind energy applications. *Journal of Applied Meteorology*, 19(9):1126–1130, 1980. [p339]

D. K. Blough, C. W. Madden, and M. C. Hornbrook. Modeling risk using generalized linear models. *Journal of Health Economics*, 18(2):153–171, 1999. [p339]

R. Chhikara and J. Folks. The inverse Gaussian distribution as a lifetime model. *Technometrics*, 19(4): 461–468, 1977. [p339]

R. S. Chhikara. *The Inverse Gaussian Distribution*. Marcel Dekker, New York, 1989. [p339]

R. S. Chhikara and J. L. Folks. Estimation of the inverse Gaussian distribution function. *Journal of the American Statistical Association*, 69(345):250–254, 1974. [p339]

P. De Jong and G. Z. Heller. *Generalized linear models for insurance data*. Cambridge University Press, Cambridge, 2008. [p339]

N. L. Johnson and S. Kotz. *Continuous Univariate Distributions, Vol. 1*. Wiley-Interscience, New York, 1970. [p339, 340]

A. G. Kallioras and I. A. Koutrouvelis. Percentile estimation in inverse Gaussian distributions. *Communications in Statistics-Simulation and Computation*, 43(2):269–284, 2014. [p339]

J. Lindsey. *rmutil: Utilities for Nonlinear Regression and Repeated Measurements Models*, 2010. URL http://www.commanster.eu/rcode.html. R package version 1.0, last changed 2010-02-15. [p339]

P. McCullagh and J. A. Nelder. *Generalized Linear Models*. Chapman & Hall/CRC, Boca Raton, Florida, 2nd edition, 1989. [p340]

J. R. Michael, W. R. Schucany, and R. W. Haas. Generating random variates using transformations with multiple roots. *The American Statistician*, 30(2):88–90, 1976. [p339, 348]

C. Pouzat. *STAR: Spike Train Analysis with R*, 2012. URL http://CRAN.R-project.org/package=STAR. R package version 0.3-7, dated 2012-10-08. [p339]

W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes in Fortran*. Cambridge University Press, Cambridge, 1992. [p339]

V. Seshadri. *The Inverse Gaussian Distribution: a Case Study in Exponential Families*. Clarendon Press, Oxford, 1993. [p339]

J. Shuster. On the inverse Gaussian distribution function. *Journal of the American Statistical Association*, 63(324):1514–1516, 1968. [p339, 341, 342]

G. Smyth and A. Verbyla. Adjusted likelihood methods for modelling dispersion in generalized linear models. *Environmetrics*, 10(6):695–709, 1999. [p339, 340]

G. K. Smyth. *invgauss: Inverse Gaussian Distribution*, 1998. URL http://www.statsci.org/s/invgauss.html. Functions for S-Plus. [p339]

G. K. Smyth. *statmod: Statistical Modeling*, 2016. URL http://CRAN.R-project.org/package=statmod. R package version 1.4.23. [p340]

M. C. Tweedie. Statistical properties of inverse Gaussian distributions I. *The Annals of Mathematical Statistics*, 28(2):362–377, 1957. [p339]

X. Wang and D. Xu. An inverse Gaussian process model for degradation data. *Technometrics*, 52(2): 188–197, 2010. [p339]

B. Wheeler. *SuppDists: Supplementary Distributions*, 2009. URL http://CRAN.R-project.org/package=SuppDists. R package version 1.1-9.1, dated 2009-12-09. [p339]

A. Whitmore, G. The inverse Gaussian distribution as a model of hospital stay. *Health Services Research*, 10(3):297, 1975. [p339]

G. Whitmore and M. Yalovsky. A normalizing logarithmic transformation for inverse Gaussian random variables. *Technometrics*, 20(2):207–208, 1978. [p349]

Göknur Giner
Bioinformatics Division, Walter and Eliza Hall Institute of Medical Research, Parkville Vic 3052, Australia; and Department of Medical Biology, University of Melbourne, Parkville Vic 3010, Australia
giner.g@wehi.edu.au

Gordon K. Smyth
Bioinformatics Division, Walter and Eliza Hall Institute of Medical Research, Parkville Vic 3052, Australia; and Department of Mathematics and Statistics, University of Melbourne, Parkville Vic 3010, Australia
smyth@wehi.edu.au

Appendix: asymptotic right tail probabilities

Here we derive an asymptotic expression for the right tail probability, $\bar{p}(q;\mu,\phi)$, when q is large. Without loss of generality, we will assume $\mu = 1$. First, we drop the $1/x$ term in the exponent of the pdf (1), leading to:

$$d(x;1,\phi) \approx \left(2\pi\phi x^3\right)^{-1/2} \exp\left(-\frac{x}{2\phi} + \frac{1}{\phi}\right)$$

for x large. Integrating the pdf gives the right tail probability as:

$$\bar{p}(q;1,\phi) \approx \exp\left(\phi^{-1}\right)(2\pi\phi)^{-1/2} \int_q^\infty x^{-3/2} \exp\left(-\frac{x}{2\phi}\right) dx$$

for q large. Transforming the variable of integration gives:

$$\bar{p}(q;1,\phi) \approx \exp\left(\phi^{-1}\right)(2\pi\phi)^{-1/2}(2\phi)^{-1/2} \int_{q/(2\phi)}^\infty x^{-3/2} \exp(-x) dx.$$

Finally, we approximate the integral using

$$\int_a^\infty x^{-3/2} \exp(-x) dx \approx (a+1)^{-3/2} \exp(-a),$$

which gives

$$\bar{p}(q;1,\phi) \approx \exp\left(\phi^{-1}\right)\pi^{-1/2}(2\phi)^{-1}\left(\frac{q}{2\phi}+1\right)^{-3/2} \exp\left(-\frac{q}{2\phi}\right)$$

and

$$\log \bar{p}(q;1,\phi) \approx \frac{1}{\phi} - 0.5\log\pi - \log(2\phi) - 1.5\log\left(\frac{q}{2\phi}+1\right) - \frac{q}{2\phi}$$

for q large.

Using DECIPHER v2.0 to Analyze Big Biological Sequence Data in R

by Erik S. Wright

Abstract In recent years, the cost of DNA sequencing has decreased at a rate that has outpaced improvements in memory capacity. It is now common to collect or have access to many gigabytes of biological sequences. This has created an urgent need for approaches that analyze sequences in subsets without requiring all of the sequences to be loaded into memory at one time. It has also opened opportunities to improve the organization and accessibility of information acquired in sequencing projects. The **DECIPHER** package offers solutions to these problems by assisting in the curation of large sets of biological sequences stored in compressed format inside a database. This approach has many practical advantages over standard bioinformatics workflows, and enables large analyses that would otherwise be prohibitively time consuming.

Introduction

With the advent of next-generation sequencing technologies, the cost of sequencing DNA has plummeted, facilitating a deluge of biological sequences (Hayden, 2014). Since the cost of computer storage space has not kept pace with biologists' ability to generate data, multiple compression methods have been developed for compactly storing nucleotide sequences (Deorowicz and Grabowski, 2013). These methods are critical for efficiently transmitting and preserving the outputs of next-generation sequencing machines. However, far less emphasis has been placed on the organization and usability of the massive amounts of biological sequences that are now routine in bioinformatics work. It is still commonplace to decompress large files and load them entirely into memory before analyses are performed. This traditional approach is quickly becoming infeasible as sequence sets swell in size, and alternative methods for storing, organizing, and analyzing sequences are needed.

A typical bioinformatics workflow begins with a set of biological sequences in one or more text files. These sequences are used as input to subsequent analysis steps, each of which generates text files as output (Schloss et al., 2011). Large workflows constructed in this manner can generate a plethora of text files, often resulting in unnecessary redundancy and disorganization. Fortunately, databases offer an organized means for storing related data, and underlie many commonly used bioinformatics software such as BLAST (Altschul et al., 1997). Nevertheless, biologists rarely use databases to curate their own sequences, in large part due to the difficulties associated with creating and accessing a database. Here I describe flexible user-friendly workflows for employing databases to efficiently analyze large sets of sequences via the R programming language.

Although R has traditionally been viewed as a statistical software, many add-on packages are available for analyzing biological sequence data. One representative, the **Biostrings** package (Pagès et al.), offers a suite of functions for reading, writing, searching, and manipulating DNA, RNA, or amino acid sequences. Sequences are stored in memory according to their corresponding `XStringSet` class, where "X" is specific to the type of sequences (e.g., "AA" for amino acid sequences). For example, a `"DNAStringSet"` can store the standard DNA bases ("A", "C", "G", or "T"), as well as ambiguity codes (e.g., "N" for any bases) and gap characters used in alignment ("-").

The **DECIPHER** package also makes use of `XStringSet` classes. However, unlike other R packages for biological sequence analysis, **DECIPHER** employs databases so that an entire sequence set does not need to simultaneously reside in memory. This enables **DECIPHER** to extend many analyses to millions of sequences without requiring extreme amounts of memory, and offers a means of handling the even more massive biological datasets of the future. The **DECIPHER** package also includes many advanced functions for oligonucleotide design (Wright et al., 2014a,b; Wright and Vetsigian, 2016), sequence alignment (Wright, 2015), and other common bioinformatics tasks. Despite its many applications, the core database functionality underpinning **DECIPHER** has not been previously described.

New users of **DECIPHER** are often unaccustomed to the use of a database to manage their own sequences. The purpose of this text is to describe the merits of this approach, and outline how sequence databases are configured by **DECIPHER** and can be used to improve analysis workflows. Sequences are stored independently within the database in a compressed format, which enables the database to be compact while maintaining fast random access to different sequences. The custom compression algorithm implemented in **DECIPHER** is compared to standard compression algorithms accessible within R. Finally, example uses of a sequence database are provided to demonstrate the power of this alternative workflow.

Merits of databases for storing biological sequences

A database, much like a spreadsheet, is an ideal way to maintain interconnected data. The concept of storing biological sequences in a relational database is not new (Xie et al., 2000), and underpins many popular bioinformatics programs and online web tools. However, end-users of these tools rarely directly employ databases for their own sequences, despite many practical advantages such as:

1. Organizational improvements:
   Databases can be arranged to minimize redundancy by maintaining an association between different columns. For example, the length of each sequence in the database can be easily added as a separate column.
2. Random access:
   Databases permit quick access to subsets of the data, without the need to seek through large text files in order to find the desired subset. For example, it is very fast to obtain the longest sequence in the database once a column with the length of each sequence has been added.
3. Concurrent users:
   A database can be queried concurrently by multiple users without interference. For example, one user can obtain the shortest sequence, while another simultaneously requests the longest sequence.
4. Reliable storage:
   Commands can be used that allow the database to revert changes in case of a mistake. **DECIPHER** workflows are designed to be non-destructive, so that the original sequence information is always preserved.

Several standard add-on packages are available to interface between R and popular database management systems (DBMSs) including MySQL, PostgreSQL, and SQLite (Ripley, 2001a). **DECIPHER** uses SQLite for a number of reasons. First, SQLite databases are flat-files that can easily be transferred between computers, or even emailed like a standard text file. Second, SQLite requires minimal setup on the part of the end-user, unlike some DBMSs. Third, support for the BLOB data type from R is currently only available from the **RSQLite** package. The BLOB type is used to store compressed sequences, which are raw (binary) type within R. Lastly, SQLite databases can contain an exceedingly large number of rows, up to 2^{64}, meaning that they are typically limited in size by the available disk space rather than the DBMS.

The use of SQLite as the sole DBMS results in a few limitations. First, users cannot be given separate access privileges, as all users are considered database administrators. Second, concurrent writes to the same database are not permitted, although concurrent reads are not a problem. These drawbacks cause few practical limitations for a typical group consisting of a few database users performing standard operations with **DECIPHER**. Furthermore, databases can be organized such that each sequencing project resides in its own table, which minimizes conflicts between users.

Anatomy of a DECIPHER database

DECIPHER uses a simple relational database schema involving two tables (Fig. 1): one that is highly "visible" to the user (named "Seqs" by default) containing information about the sequences, and a second "hidden" table (named "_Seqs") for storing compressed sequences and, if applicable, their corresponding quality scores. The tables are connected by a shared primary key, named "row_names", that enables fast lookup between the two tables. This split table design substantially increases access speed over using a single table, as the table being queried does not include any sequences, which are often large in size and can slow access to other data. Use of separate tables within the same database is provided by the argument `tblName` in each function.

Sequences are imported with the `Seqs2DB` function which supports three popular file formats as well as in-memory `XStringSet` objects. The destination table is automatically populated with appropriate text columns containing information stored in the file. For example, sequences imported from FASTA files store each sequence's record name in a column named "description". FASTQ files are imported similarly, but also store the quality information corresponding to each sequence in gzip compressed format. For GenBank files, the "description" column is obtained from the DEFINITION field and other fields can be imported as desired using the `fields` argument of `Seqs2DB`. By default, the ACCESSION and ORGANISM fields are imported as additional columns named "accession" and "rank" in the database.

All **DECIPHER** databases require an "identifier" to be specified during import. The identifier column is used extensively by **DECIPHER** functions, and is the recommended way to delineate groups of sequences. Common ways to identify sequences include: the file they were imported from

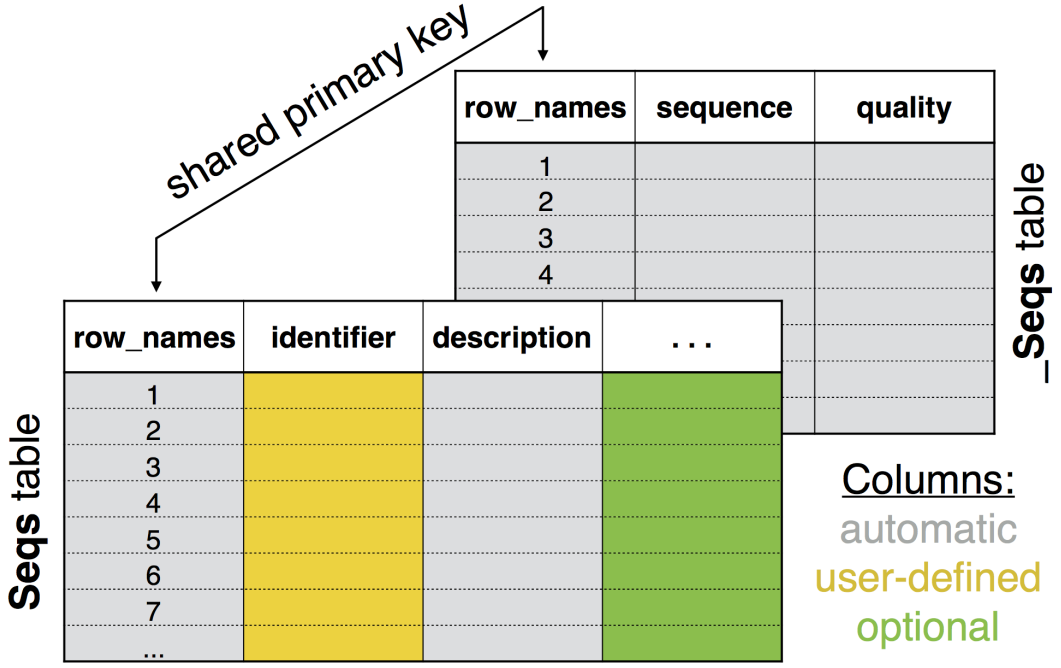


Figure 1: Database schema used by **DECIPHER**. Two tables are created when importing sequences with `Seqs2DB`: one that is highly visible to the user (named "Seqs" by default) and a second (named "_Seqs") that is largely hidden. The tables are connected by a shared primary key ("row_names") that enables fast lookup between the two tables. Columns containing the sequence descriptors, compressed sequences, and compressed quality scores (if applicable) are automatically generated. The user must specify an "identifier" during import, which can be changed later through a variety of methods. Additional columns of data may be added to the database using the `Add2DB` function.

(e.g., "file1", "file2", etc.), the cluster they belong to in a phylogeny (e.g., "cluster1", "cluster2", etc.), or the name of the organism from which the sequence originated (e.g., "E. coli", "Yeast", etc.). It is also possible to simply provide an empty character string (i.e., "") as the identifier during import, and then set the identifier later. **DECIPHER** includes several functions to help with identifying the sequences after they are imported. For example, the `IdClusters` function can assign phylogenetic cluster numbers. The `IdentifyByRank` function can parse the "rank" column from an imported GenBank file to identify sequences according to a given taxonomic rank (e.g., "species").

DECIPHER uses R's connection interface (Ripley, 2001b) to read files incrementally during import. Files compressed with gzip, bzip2, xz, or lzma compression are automatically detected and read appropriately without user intervention. The user can also provide a URL instead of a file path, which enables sequences to be imported from "http" or "ftp" sources without first needing to save the file locally. In the case of URLs, only uncompressed text files or files with gzip compression are supported. Notwithstanding this limitation, reading files directly from online sources, such as NCBI repositories, is often preferable to downloading files locally before importing them into a database as it prevents unnecessary redundancy. An example of importing a compressed GenBank file from online is shown below:

```
> library(DECIPHER)
>
> # specify the input file and database location
> gbk_file <- "ftp://ftp.ncbi.nlm.nih.gov/genomes/H_sapiens/RNA/rna.gbk.gz"
> db_file <- "~/Desktop/SeqsDB.sqlite"
> dbConn <- dbConnect(SQLite(), db_file)
>
> # import the sequences from online
> Seqs2DB(seqs = gbk_file,
+         type = "GenBank",
+         dbFile = dbConn,
+         identifier = "Human_mRNA")
162916 total sequences in table Seqs.
Time difference of 176.19 secs
```

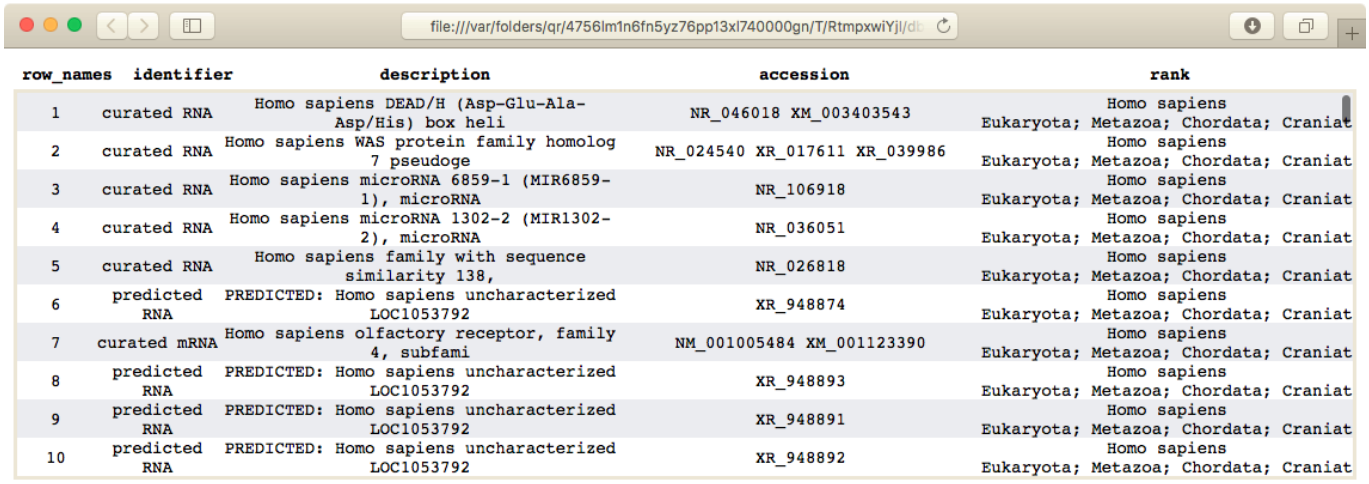


Figure 2: Screenshot of viewing a database table in a web browser using the `BrowseDB` function. Metadata information such as accession numbers and taxonomy are automatically imported from GenBank sequence files into separate table columns. The value in the "identifier" column controls sequence groupings in many **DECIPHER** functions, and can be set during or after import by a variety of methods. For easier viewing, the text in each field is truncated at a maximum of 50 characters by default.

The imported file contains known and predicted human RNA sequences. At this point it is useful to reset the values in the identifier column so that they can be referenced in downstream analyses. In this particular example, the prefix of the accession number can be parsed into one of four values, indicating whether the sequence is a predicted or hand-curated RNA or messenger RNA (mRNA) sequence. This information is updated in the database using the `Add2DB` function, which, like many other **DECIPHER** functions, displays the associated SQL commands when the input argument `verbose` is `TRUE`. `Add2DB` will add or update table columns in accordance with the column names and row names in a `"data.frame"` input. For example, in this case the values of identifier in the input `"data.frame"` are added to the rows with corresponding "row_names" in the database. To view the results of this modification, the database table can be displayed in a web browser with the `BrowseDB` function (Fig. 2).

```
> # reset the 'identifier' column based on the prefix of the accession number
> x <- dbGetQuery(dbConn, "select accession from Seqs")
> id <- substring(x$accession, 1, 2)
> id <- c(`NM` = "curated mRNA",
+         `NR` = "curated RNA",
+         `XM` = "predicted mRNA",
+         `XR` = "predicted RNA")[id]
> Add2DB(data.frame(identifier = id), dbConn)
Expression:
update or replace Seqs set identifier = :identifier where row_names =
:row_names

Added to table Seqs:  "identifier".
Time difference of 2.06 secs

> BrowseDB(dbConn, limit = 1000)
```

The nbit compression format for nucleotides

Many compression algorithms have been proposed for storing nucleotide sequences, some of which are specific to the FASTQ file format (Deorowicz and Grabowski, 2013). The most common compression method is gzip, owing to its reasonable compression ratio and high decompression rate. However, since gzip is a generalized method, it may be possible to obtain better compression ratios by using algorithms specific to DNA sequences. In particular, reference-based compression is generally preferable when a reference sequence is available (Jones et al., 2012). In re-sequencing projects a reference genome

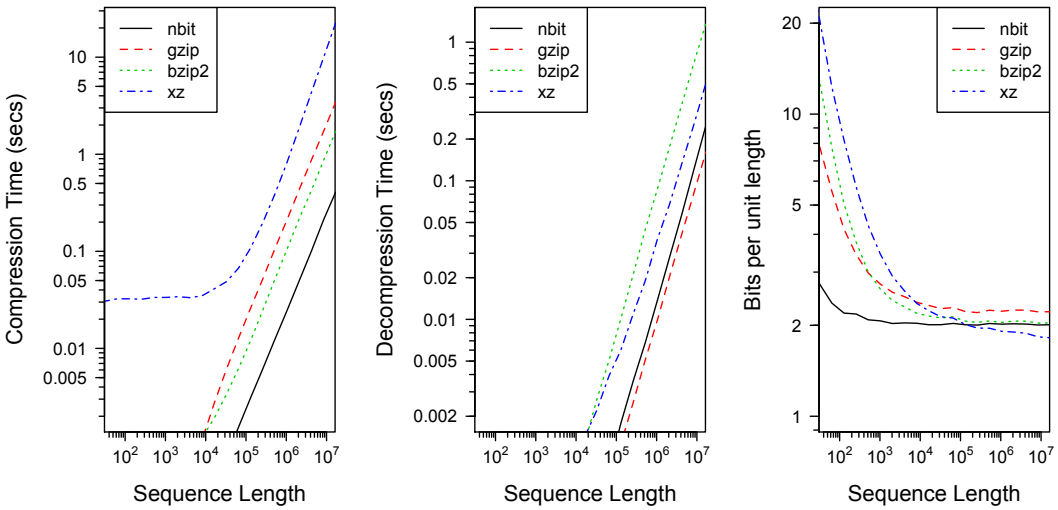


Figure 3: Comparison between different lossless compression algorithms on random subsequences of Human Chromosome II (HG18). **DECIPHER**'s custom **nbit** compression exhibited substantially faster compression rates than the other methods. The **nbit** algorithm offered a better compression ratio than any of the other methods for sequences less than about 100,000 nucleotides, beyond which xz compression provided more compaction, albeit at a substantially slower compression rate. The average of 100 replicates is shown for all methods, gzip (v1.2.8), bzip2 (v1.0.6), and xz (v5.0.7), computed using a single processor.

is always available, whereas it is often unavailable for new sequencing projects. In either case, compression of a file containing many similar sequences allows redundancy to be exploited to a greater extent than independently compressing sequences.

Several considerations were taken into account when designing a compression method for **DECIPHER**. First, the method must work well with a wide variety in the number of sequences and their lengths. Second, in order to allow random access and deposition, each sequence is stored independently in the database, ruling out exploiting redundancy between sequences. Third, compression and decompression rates were prioritized over achieving the maximal possible compression ratio. Fourth, support for all possible characters in the DNA and RNA alphabets was desired, in particular the gap character that is largely neglected by most DNA-specific compression formats. These considerations led to the development of the **nbit** compression method for DNA or RNA sequences. Compression with **nbit** uses a combination of 2-bit encoding for gapless bases (i.e., A=00, C=01, G=10, T=11), and 3-bit encoding for gappy regions. Although this encoding was inspired by prior work (Wandelt et al., 2014), **DECIPHER** uses a unique implementation that is customized to the package's goals.

In principle, the maximum achievable compression rate is 2-bits per base with this encoding, which is the theoretical limit (i.e., Shannon entropy) for four randomly drawn characters in equal proportions. However, DNA sequences contain appreciable information and are non-random in their construction, which permits additional compression to be achieved. To this end, the **nbit** algorithm incorporates runs of a single base or ambiguity codes (e.g., "NNN"), which are frequent in some sequences. Furthermore, long DNA sequences often contain exact repeats or reverse complement repeats, which can be stored compactly by referencing their prior occurrence. Finally, the **nbit** format includes a variable-sized cyclic redundancy check to ensure data integrity.

The **nbit** compression and decompression algorithms were implemented in the **DECIPHER** function `Codec`, which interconverts between character (uncompressed) and binary (compressed) data. In comparison to the three generalized compression methods available in R, the **nbit** algorithm exhibits a substantially faster compression rate, as shown in Figure 3. Compression sizes are generally better than those of the other methods, except for sequences longer than about 100,000 nucleotides where xz compression results in more compaction at the expense of substantially longer compression times. The user may also specify to use more than one processor with **nbit**, in which case the `Codec` function will compress/decompress sequences in parallel. Furthermore, the `Codec` function will automatically fall back to gzip compression when the sequences are incompressible with **nbit**, as in the case of amino acid sequences.

Figure 4: Sequences returned from a database query are displayed in a web browser using the `BrowseSeqs` function. The mRNA sequences shown here all match the query "transcription factor" and are named by their accession number(s). Sequence positions are colored according to their DNA base (A, C, G, or T) and wrapped at 100 nucleotides.

Example workflow with DECIPHER

One of the major advantages of using a sequence database is the ability to quickly access subsets of the sequences matching certain criteria. The `SearchDB` function can be used to easily build flexible queries and obtain the sequences meeting those specifications. `SearchDB` supports several common SQL clauses, including 'LIMIT', 'OFFSET', 'ORDER BY', and 'WHERE'. If unspecified, `SearchDB` can automatically detect the type of sequences (DNA, RNA, or AA) returned from the search. In addition, it can be used to quickly count the number of sequences matching a query, name the sequences based on the value in a specific table column, remove gaps ("-") from sequences, or replace characters not present in the specified sequence alphabet. As an example, the command below will find all of the "curated mRNA" sequences with "transcription factor" in their "description" and name the sequences by their accession number. The sequences can then be viewed in a web browser using the `BrowseSeqs` function, as shown in Figure 4.

```
> dna <- SearchDB(dbConn,
+                 identifier = "curated mRNA",
+                 nameBy = "accession",
+                 clause = "description like '%transcription factor%'")
Search Expression:
select accession, _Seqs.sequence from Seqs join _Seqs on Seqs.row_names =
_Seqs.row_names where _Seqs.row_names in (select row_names from Seqs where
identifier is "curated mRNA" and description like '%transcription factor%')

DNAStringSet of length: 437
Time difference of 0.44 secs

> BrowseSeqs(dna, colWidth = 100)
```

Several **DECIPHER** functions make use of `SearchDB`'s `limit` argument to extract batches of sequences. The `limit` argument can be either a single numeric value indicating the maximum number of sequences to return, or a character string specifying two numbers separated by a comma giving the offset and limit. For example, the `IdLengths` function makes use of this feature to efficiently compute the lengths of all of the sequences in a database table. This length information is then added to the database in batches using the `Add2DB` function. A similar workflow is used by many **DECIPHER** functions so that all of the sequences do not have to be kept in memory simultaneously. An example of using the `IdLengths` function is shown below:

```
> l <- IdLengths(dbConn, add2tbl = TRUE)
  |======================================================================| 100%
Lengths counted for 162916 sequences.
Added to Seqs:  "bases", "nonbases", and "width".
```

```
Time difference of 26.15 secs
```

Rather than using offset and limit values, several **DECIPHER** functions make use of the user-specified identifier to split the sequences into groups. For example, the `IdConsensus` function will create a single consensus sequence for the sequences corresponding to each identifier in a table, by accessing one identifier's sequences at a time. It is also possible to construct more sophisticated queries of the database using the `clause` argument of the `SearchDB` function. The clause will be appended to the query after the keyword 'WHERE'. Below is an example of using the `clause` argument to retrieve the longest 'curated RNA' sequence in the table, which is 91,671 nucleotides in length.

```
> SearchDB(dbConn,
+          identifier = "curated RNA",
+          clause = "bases = (select max(bases) from Seqs
+                    where identifier is 'curated RNA')")
Search Expression:
select row_names, sequence from _Seqs where row_names in (select row_names
from Seqs where identifier is "curated RNA" and bases = (select max(bases)
from Seqs where identifier is 'curated RNA'))

DNAStringSet of length: 1
Time difference of 0.25 secs

  A DNAStringSet instance of length 1
    width seq                                                 names
[1] 91671 AGGCAGAACGGTCGCCGCGTCGC...ATGAAAACTATGAAAACTGACTA 73201
```

As the above example demonstrates, a knowledge of basic SQL syntax is helpful to harness the full potential of **DECIPHER** for analyzing big biological sequence data in R. Nevertheless, it is still possible to automatically generate a wide variety of complex queries using only the input arguments of **DECIPHER**'s database functions. For example, the `DB2Seqs` function can be used to export a large number of sequences meeting certain criteria from the database. The sequences are exported in FASTA format, or FASTQ format if corresponding quality scores are available. The code below exports a gzip compressed FASTA file containing all of the "predicted mRNA" sequences, ordered by their length, and with each sequence record named by its accession number.

```
> out_file <- "~/Desktop/seqs.fas.gz"
> DB2Seqs(out_file,
+         dbConn,
+         identifier = "predicted mRNA",
+         nameBy = "accession",
+         orderBy = "bases",
+         compress = TRUE)
  |======================================================================| 100%

Wrote 61051 sequences.
Time difference of 52.22 secs
```

Access times vary greatly depending on where the database is stored. If the database is small enough, very fast read and write speeds can be obtained by keeping the entire database in memory. For moderate to large sets of sequences it is necessary to store the database on a drive, as demonstrated in the examples above. Note that access speed can be delayed considerably when the database is kept in a shared location, such as on a networked drive. Independently of the database's location, it is important to disconnect after finishing the database session. This would permanently destroy an in-memory database, but only reversibly closes a connection to a drive-based database, as shown here:

```
> dbDisconnect(dbConn)
[1] TRUE
```

Conclusions

DECIPHER is a versatile R package for the curation of biological sequence sets. Its `Seqs2DB` function constructs a database that can be used to efficiently store DNA, RNA, or amino acid sequences in compressed format. For DNA and RNA, **DECIPHER** employs a custom compression algorithm, called

nbit, that enables fast compression and decompression at a reasonable compression ratio. Using a database, it is possible to construct non-destructive workflows that handle sequences in batches so that they do not need to be kept in memory simultaneously. The `SearchDB` function can be used to automatically generate complex queries that return sequences from a database table. Collectively, **DECIPHER** functions make it possible to process millions of biological sequences with relative ease. This capability will only become more useful as improvements in sequencing power continue to outpace growth in memory capacity.

Bibliography

S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997. [p352]

S. Deorowicz and S. Grabowski. Data compression for sequencing data. *Algorithms for Molecular Biology*, 8(1):1–13, 2013. [p352, 355]

E. C. Hayden. Technology: The $1,000 genome. *Nature*, 507(7492):294–295, Mar. 2014. [p352]

D. C. Jones, W. L. Ruzzo, X. Peng, and M. G. Katze. Compression of next-generation sequencing reads aided by highly efficient de novo assembly. *Nucleic Acids Research*, 40(22):e171–e171, Dec. 2012. [p355]

H. Pagès, P. Aboyoun, R. Gentleman, and S. DebRoy. *Biostrings: String objects representing biological sequences, and matching algorithms*. R package version 2.38.0. [p352]

B. D. Ripley. Using Databases with R. *R News*, 1(1):18–20, Jan. 2001a. [p353]

B. D. Ripley. Connections. *R News*, 1(1):16–17, Jan. 2001b. [p354]

P. D. Schloss, D. Gevers, and S. L. Westcott. Reducing the Effects of PCR Amplification and Sequencing Artifacts on 16S rRNA-Based Studies. *PloS one*, 6(12):e27310, Dec. 2011. [p352]

S. Wandelt, M. Bux, and U. Leser. Trends in genome compression. *Current Bioinformatics*, 2014. [p356]

E. S. Wright. DECIPHER: harnessing local sequence context to improve protein multiple sequence alignment. *BMC Bioinformatics*, 16(1):1–14, Sept. 2015. [p352]

E. S. Wright and K. H. Vetsigian. DesignSignatures: a tool for designing primers that yields amplicons with distinct signatures. *Bioinformatics*, Jan. 2016. [p352]

E. S. Wright, L. S. Yilmaz, A. M. Corcoran, H. E. Okten, and D. R. Noguera. Automated design of probes for rRNA-targeted fluorescence in situ hybridization reveals the advantages of using dual probes for accurate identification. *Applied and environmental microbiology*, 80(16):5124–5133, July 2014a. [p352]

E. S. Wright, L. S. Yilmaz, S. Ram, J. M. Gasser, G. W. Harrington, and D. R. Noguera. Exploiting extension bias in polymerase chain reaction to improve primer specificity in ensembles of nearly identical DNA templates. *Environmental Microbiology*, 16(5):1354–1365, May 2014b. [p352]

G. Xie, R. DeMarco, R. Blevins, and Y. Wang. Storing biological sequence databases in relational form. *Bioinformatics*, 16(3):288–289, Mar. 2000. [p353]

Erik S. Wright
University of Wisconsin - Madison
330 N Orchard St, Madison WI 53715 USA
eswright@wisc.edu

R Packages to Aid in Handling Web Access Logs

by Oliver Keyes, Bob Rudis, Jay Jacobs

Abstract Web access logs contain information on HTTP(S) requests and form a key part of both industry and academic explorations of human behaviour on the internet. But the preparation (reading, parsing and manipulation) of that data is just unique enough to make generalized tools unfit for the task, both in programming time and processing time which are compounded when dealing with large data sets common with web access logs. In this paper we explain and demonstrate a series of packages designed to efficiently read in, parse and munge access log data, allowing researchers to handle URLs and IP addresses easily. These packages are substantially faster than existing R methods - from a 3-500% speedup for file reading to a 57,000% speedup in URL parsing.

Introduction

The rise of the World Wide Web has made it dramatically easier to access and transfer data, and R boasts abundant functionality when it comes to taking data *from* the web. Base R itself has simple file downloading and page reading capabilities, through the `download.file` and `readLines` functions, and additional functionality is made available for handling web-accessible data through packages such as **httr** (Wickham, 2015).

Data *on* the web is not, however, the only kind of web data that interests researchers; web traffic is, in and of itself, an interesting data source. Access logs–records of connections between users and a web server–are an asset and resource for people studying everything from user behaviour on the internet (Halfaker et al., 2014), to website performance (Ryckbosch and Diwan, 2014), to information security (Bhingarkar and Shah, 2015).

An example scenario for those users is the need to be able to read in access logs, parse the URLs to extract relevant metadata about the access requests, geolocate the people making those requests to understand how behaviour and desired content vary between populations.

As a statistically-oriented programming language, R is commonly used by these same researchers for data analysis, testing and reporting. However, the general-purpose functions within R require significant customization before they can handle the scenario above. Additionally, the off-the-shelf functions are inefficient for these tasks, especially when scaling to the large data sets that are common with web access logs. In this article we review the use cases for particular operations over web data, the limitations in base R when it comes to performing those operations, and a suite of R packages designed to overcome them: reading access logs in (**webreadr**), manipulating URLs (**urltools**), manipulating IP addresses (**iptools**), and direct IP geolocation (**rgeolocate**).

Reading access logs

The first task with any data analysis is to read the data into R. With access logs this is slightly complicated by the fact that there is no one standard for what a log should look like; instead, there are multiple competing approaches from different software platforms and eras. These include the Common Log Format (CLF), the confusingly-named Combined Log Format, and formats used by individual, commonly-used software platforms - such as the custom format for the Squid internet caching software, and the format used by Amazon Web Services (AWS).

One difference between formats can easily be shown by looking at how timestamps are represented:

Table 1: Timestamps in Common Access Log Formats

Log Type	Timestamp Columns	Timestamp Format
Common Log Format	1	10/Oct/2000:13:55:36 -0700
Combined Log Format	1	26/Apr/2000:00:23:48 -0400
Squid	1	1286536309.450
AWS	2	2014-05-23 01:13:11

With four log types, we have three different timestamp formats, and `timestamp` is only one of the many columns that could appear. These logs also vary in whether they specify quoting fields (or sanitising unquoted ones), the columns they contain and the data each column contains in turn.

Base R does not have a way of easily reading in any one of those formats, let alone all of them (understandably, given its statistical orientation). There is also the **ApacheLogProcessor** (Mendonca, 2015) package, but (although more dedicated than base R) it is only capable of reading Common Log Format files (although it does have novel features such as optional parallel processing for the data cleanup tasks).

To make reading access logs into R as easy as possible we created the **webreadr** (Keyes, 2015) package. This contains user-friendly equivalents to `read.table` for each type of log, detecting the fields that should appear, converting the timestamps into POSIX objects, and merging fields or splitting fields where necessary. The package contains four core functions, one for each form of log: `read_clf` for the Common Log Format, `read_combined` for the Combined Log Format, and `read_squid` and `read_aws` for Squid and AWS formats respectively. Each one abstracts away the complexity of specifying column names and formats, and instead allows a researcher to read a file in with a minimal amount of work: the only parameters that need to be specified are the path to the file, and whether the file has column headers.

As the name suggests, it is built not on top of base R but on top of the **readr** (Wickham and Francois) package, allowing us to take advantage of substantial speed improvements that package's base functions have over base R (**ApacheLogProcessor**, in contrast, relies on those same base functions). These improvements can be seen in the visualisation below, which uses **microbenchmark** (Mersmann, 2014) to compare 100 reads of a 600,000-line "squid" formatted file with webreadr to the same operation performed in base R:

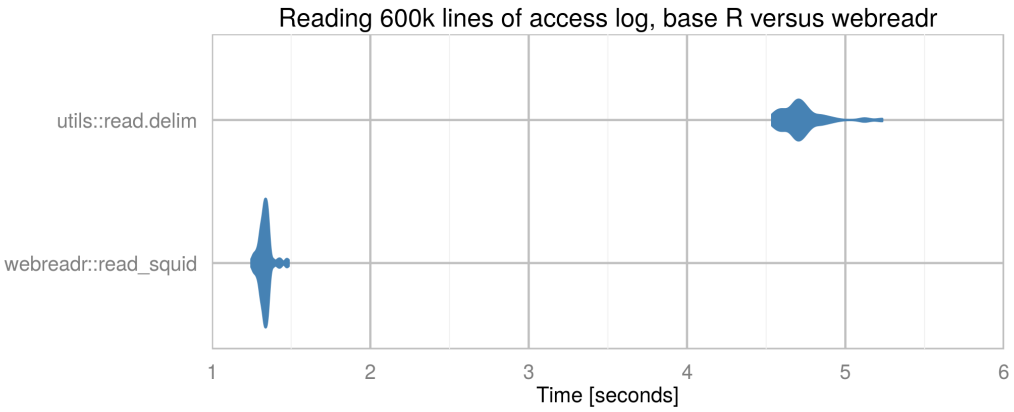


Figure 1: Results of microbenchmark run: read_squid versus base-R equivalent code

As this plot shows, **webreadr** is consistently 3.5-6 times faster than the equivalent base R functionality, and, as explained above, is also far simpler to use. **ApacheLogProcessor** was not benchmarked for the simple reason that it is built on the base R functionality tested, and so would be duplicative.

Decoding and parsing URLs

URLs are commonplace in access logs, describing both the web asset or page that the user requested, and the page the user came from. These fields are usually named url and referer respectively.

Decoding

Both values can be percent-encoded, allowing them to include characters that are valid but reserved by the URL specification as having special meanings ("reserved characters"). A '#' symbol, for example, is encoded as '%23': a percentage symbol, followed by a unique numerical value for that character.

The encoding of reserved characters is useful, since it means that URL paths and queries can contain a vast array of values - but it makes data analysis tougher to do. Examples of common data analysis or cleaning operations that become more difficult are:

1. **Aggregation**. Aggregating URLs together is useful to identify, for example, the relative usage and popularity of particular pages in your data - but it becomes tougher if encoding is inconsistent, because two URLs could hold the same value but *look* very different.
2. **Value selection**. With text-based data, regular expressions are a common way of filtering or selecting entries that meet particular conditions, but things become fuzzy when you have to look not just for particular characters (a space, say) but also the encoded values (%20).

3. **Exploratory data analysis** (EDA). EDA is a common initial step to investigate a data set, examining the variables and values it contains and whether they meet a researcher's expectations - but on a practical basis it becomes difficult when the values aren't human-readable.

The solution is to be able to consistently decode URLs, which makes URL-based data far easier to analyse. Base R contains the function `URLdecode` for this purpose, but, as it is neither vectorised nor based on compiled code, it can be extremely slow over large datasets.

To solve this common problem in analysing request logs, the **urltools** (Keyes et al., 2015a) package was created. This contains a function, `url_decode`, which decodes URLs and relies on vectorised, compiled code to do so. Benchmarking the two approaches against each other shows that the **urltools** implementation is approximately 60-70 times faster over large datasets. Again using **microbenchmark**, if we compare the vectorised decoding of 1,000,000 URLs with **urltools** against a `URLdecode vapply` loop, we see a 60-70 times speed improvement:

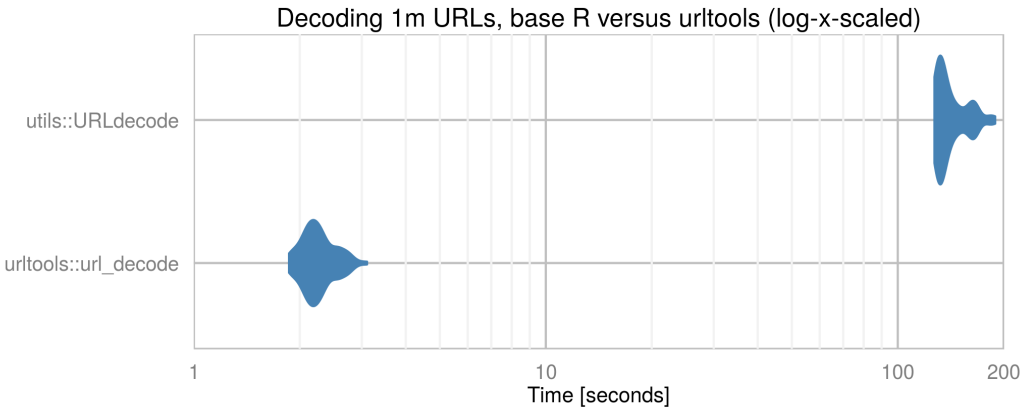


Figure 2: Results of microbenchmark run: `url_decode` versus base-R equivalent code

Parsing

The standard for URLs (Berners-Lee et al., 1994) divides them into a heirarchical sequence of components - the scheme ('`http`'), host ('`en.wikipedia.org`'), port ('`800`'), path ('`wiki/Main_Page`') and search-part, or query string ('`action=edit`'). Together, these make up a URL ('`http://en.wikipedia.org:800/wiki/Main_Page?action=edit`').

Parsing URLs to isolate and extract these components is a useful ability when it comes to exploring request logs; it lets a researcher pick out particular schemes, paths, hosts or other components to aggregate by, identifying how users are behaving and what they are visiting. It makes anonymising data - by removing, for example, the parameters or path, which can contain relatively unique information - easier.

Base R does not have native code to parse URLs, but the **httr** package (Wickham, 2015) contains a function, `parse_url`, designed to do just that. Built on R's regular expressions, this function is not vectorised, does not make use of compiled code internally, and produces a list rather than data.frame, making looping over a set of URLs to parse each one a time-consuming experience. This is understandable given the intent behind that function, which is to decompose individual URLs within the context of making HTTP requests, rather than to analyse URLs *en masse*. Similarly, the **XML** (Lang and the CRAN Team, 2016) package has `parseURI`; C-based, this time, but both dependent on the `libxml` library and, similarly, not vectorised.

urltools contains `url_parse` - which does the same thing as the equivalent **httr** functionality, but in a vectorised way, relying on compiled code, and producing a `data.frame`. Within the context of parsing and processing access logs, this is far more useful, because it works efficiently over large sets: **httr**'s functionality, which was never designed with vectorisation in mind, does not. Indeed, benchmarking showed that `url_parse` is approximately 570 times faster than **httr**'s equivalent function:

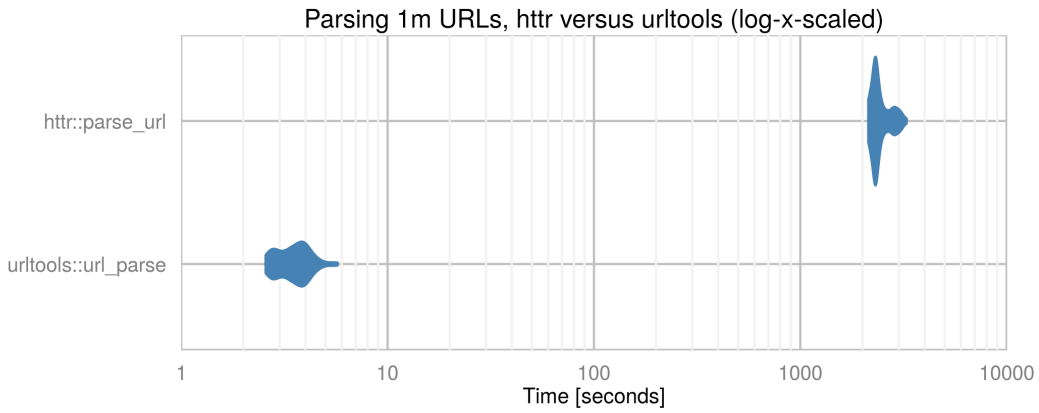


Figure 3: Results of microbenchmark run: `url_parse` versus httr's equivalent code

A vector of URLs passed into `url_parse` produces a data.frame, with one column for each of the IETF-supported components, and empty strings representing components that could not be found. Additionally, influenced by the style of the **lubridate** package (Grolemund and Wickham, 2011), **urltools** contains functions to get or *set* individual components:

```
# Load urltools and construct a URL
library(urltools)
url <- "http://www.google.com/"

# Get the scheme
scheme(url)

#> [1] "http"

# Set the scheme and observe the modification to the resulting URL
scheme(url) <- "https"
url

#> [1] "https://www.google.com/"
```

As a result of this functionality, **urltools** makes URL manipulation faster, easier and far more accessible, reducing the burden associated with munging access logs or similar datasets and allowing a researcher to get to the statistical analysis faster.

IP manipulation

Access logs also contain IP addresses - unique numeric values that identify a particular computer or network in the context of the internet. Working with these values allows an analyst to validate their data (by checking for false or spoof addresses) and is a necessary prerequisite to the use of some IP geolocation systems (covered later), and extract a limited amount of metadata from the values themselves.

iptools (Rudis and Keyes, 2015), based around the *Boost ASIO* C++ library (boo), is a package designed for this kind of IP manipulation (and more). Built around combined code, it is extremely fast (for that code that does not rely on internet connections) and boasts a range of features.

One of the most crucial is `ip_classify`, which, when provided with a vector of IP addresses, identifies whether they follow the IPv4 or IPv6 standard. As a side-effect of this, it can be used to identify if IP addresses are invalid, or spoofed, prior to further work based on the assumption that they are correct:

```
# Load iptools and construct a vector of IPs
library(iptools)
ip_addresses <- c("192.168.0.1", "2607:f8b0:4006:80b::1004", "Chewie")
ip_classify(ip_addresses)

#> [1] "IPv4"    "IPv6"    "Invalid"
```

Along with this general tool there are also functions for ascertaining specific facts about IP addresses, returning logical (`TRUE` or `FALSE`) values. These are `is_valid`, `is_ipv4`, `is_ipv6`, which are built on top of `ip_classify`, and `is_multicast`, which identifies if an IP address is *multicast* - designed to point to multiple machines.

iptools also contains (for similar purposes) code to identify the actual client's IP address. Access requests usually contain not only an IP address but a *X-Forwarded-For* field - a field identifying which *other* IP addresses the request passed through, if the user who made the request is using some kind of proxy. If a user *has* used a proxy, the contents of the IP address field won't actually be them - it will be the last proxy the request went through before getting to the server logging the requests. The actual IP address of the user will instead be the earliest value of the X-Forwarded-For field.

The solution is to be able to identify the 'real' IP address, by checking:

1. Whether the X-Forwarded-For field contains any values;
2. Extracting the earliest non-invalid IP address in that field's values if so, and the contents of the IP address field if not.

With the `xff_extract` function, you can do just that:

```
ip_address <- "192.168.0.1"
x_forwarded_for <- "foo, 193.168.0.1, 230.98.107.1"
xff_extract(ip_address, x_forwarded_for)

#> [1] "192.168.0.1"
```

This returns the IP address field value if X-Forwarded-For is empty, and otherwise splits the X-Forwarded-For field and returns the earliest valid IP address. It is fully vectorised and highly useful for analysis predicated on IP addresses being valid, such as geolocation - without this kind of resolution, what you're actually geolocating might be your own servers.

Other operations supported by **iptools** include the conversion of IP addresses from their standard *dotted-decimal* form to a numeric form, the extraction of IP ranges (and their contents), resolving IP addresses to hostnames, and a series of datasets covering the IPv4 registry and port database.

The limitation of the package is that some operations do not yet support IPv6 (since they require the storage of numbers bigger than R can currently handle).

Geolocation

As a side-effect of how IP addresses tend to be assigned - in geographic blocks, to individual machines or to local networks - they can be used to geolocate requests, identifying where in the world the request came from, sometimes down to the level of individual post codes or pairs of latitude/longitude coordinates.

This is tremendously useful in industry, where the geographic reach of a service has substantial implications for its viability and survivability, and in academia, where the locality of internet-provided information and the breadth of internet access are active concerns and areas of study (Sen et al., 2015).

Many services and databases exist for extracting geographic metadata from IP addresses. One of the most common is the service provided by MaxMind, which has both proprietary and openly-licensed databases, in binary and comma-separated formats. The free databases have been used by various web APIs, which makes the data they contain accessible from R. Unfortunately, dependence on web APIs means that handling large numbers of IP addresses can be very slow (they tend to be designed to only accept one IP address at a time, and may contain throttling beyond that) and has privacy concerns, since it essentially means sending user IP addresses to a third party. And even without these issues, there are no wrappers for those APIs available on CRAN: users have to write their own.

With these concerns in mind, we wrote the **rgeolocate** (Keyes et al., 2015b) package. Through **httr** this contains convenient, vectorised bindings to various web services that provide geographic metadata about IP addresses. More importantly, using the **Rcpp** package to integrate C++ and R, **rgeolocate** also features a direct, compiled binding to the MaxMind API. This means that local binary databases can also be queried, which is far faster and more robust than web-based equivalents and avoids the privacy concerns associated with transmitting users' IP addresses externally.

The MaxMind API requires a paid or free binary database - one of which, for country-level IP resolution, is included in **rgeolocate** - and allows you to retrieve the continent, country name or ISO code, region or city name, tzdata-compatible timezone, longitude, latitude or connection type of a particular IP address. Multiple fields can be selected (although which are available depends on the type of database used), and results are returned in a data.frame:

```
# Load rgeolocate
library(rgeolocate)

# Find the rgeolocate-provided binary database
geo_file <- system.file("extdata","GeoLite2-Country.mmdb", package = "rgeolocate")

# Put together some example IP addresses
ip_addresses <- c("174.62.175.82", "196.200.60.51")

# Geolocate
maxmind(ip_addresses, geo_file, fields = c("continent_name", "country_code"))

#>   continent_name country_code
#> 1  North America           US
#> 2         Africa           ML
```

Direct speed comparisons aren't possible, since the functionality it provides is not, to the authors' knowledge, replicated in other R packages, but it is certainly faster and more secure than internet-dependent alternatives.

Conclusions and further work

In this research article we have demonstrated a set of tools for handling access logs during every stage of the data cleaning pipeline - reading them in with **webreadr**, decoding, manipulating and extracting value from URLs with **urltools**, and retrieving geographic metadata from IP addresses with **iptools** and **rgeolocate**. We have also demonstrated the dramatic speed improvements in using these tools in preference to existing methods within R.

In combination, this makes for an incredibly powerful set of tools for analysing web data, providing functionality and economies of scale not previously available in R and making R a first-class environment for web data analysis. In the introduction, we gave an example of a common workflow for researchers dealing with these sorts of logs: knitting the examples above together, we can see that the tasks have gone from inefficient to efficient (in the cases of data reading and URL manipulation) and from impossible to possible in the case of IP geolocation.

Further work - integrating more sources of geolocation information within **rgeolocate**, supporting UTF-8 URL decoding within **urltools**, and increasing IPv6 support in **iptools** - would make these packages even more useful to Human-Computer Interaction researchers and other specialists who rely on access logs as primary data sources. Possible sources of performance improvements include Even without that functionality, however, this suite of packages is a dramatic improvement upon the status quo.

Acknowledgements

This paper would not have been possible without the support of Margret Wander, Adam Hyland and Penelope Hopkins, and the copyediting and commentary by Laurent Gatto, Brandon Hurr and Hadley Wickham. In addition, we would like to thank Toby Negrin for inspiring us to write a paper on these packages, rather than just the packages themselves.

The code for the benchmarking included in the figures can be found in the git repository for the paper, and is MIT-licensed.

Bibliography

Boost C++ Libraries. URL http://www.boost.org. http://www.boost.org. [p363]

T. Berners-Lee, L. Masinter, and M. McCahill. Uniform resource locators (url). RFC 1738, December 1994. URL https://tools.ietf.org/html/rfc3986. [p362]

A. S. Bhingarkar and B. D. Shah. A survey: Securing cloud infrastructure against EDoS attack. In *Proceedings of the International Conference on Grid Computing and Applications (GCA)*, page 16. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2015. [p360]

G. Grolemund and H. Wickham. Dates and times made easy with lubridate. *Journal of Statistical Software*, 40(3):1–25, 2011. URL http://www.jstatsoft.org/v40/i03/. [p363]

A. Halfaker, O. Keyes, D. Kluver, J. Thebault-Spieker, T. T. Nguyen, K. Shores, A. Uduwage, and M. Warncke-Wang. User session identification based on strong regularities in inter-activity time. *CoRR*, abs/1411.2878, 2014. URL http://arxiv.org/abs/1411.2878. [p360]

O. Keyes. *webreadr: Tools for Reading Formatted Access Log Files*, 2015. URL http://CRAN.R-project.org/package=webreadr. R package version 0.3.0. [p361]

O. Keyes, J. Jacobs, M. Greenaway, and B. Rudis. *urltools: Vectorised Tools for URL Handling and Parsing*, 2015a. URL http://CRAN.R-project.org/package=urltools. R package version 1.3.2. [p362]

O. Keyes, D. Schmidt, D. Robinson, I. Maxmind, and P. Gloor. *rgeolocate: IP Address Geolocation*, 2015b. URL http://CRAN.R-project.org/package=rgeolocate. R package version 0.5.0. [p364]

D. T. Lang and the CRAN Team. *XML: Tools for Parsing and Generating XML Within R and S-Plus*, 2016. URL https://CRAN.R-project.org/package=XML. R package version 3.98-1.4. [p362]

D. S. Mendonca. *ApacheLogProcessor: Process the Apache Web Server Log Combined Files*, 2015. URL http://CRAN.R-project.org/package=webreadr. R package version 0.1.5. [p361]

O. Mersmann. *microbenchmark: Accurate Timing Functions*, 2014. URL http://CRAN.R-project.org/package=microbenchmark. R package version 1.4-2. [p361]

B. Rudis and O. Keyes. *iptools: Manipulate, Validate and Resolve IP Addresses*, 2015. URL http://CRAN.R-project.org/package=iptools. R package version 0.3.0. [p363]

F. Ryckbosch and A. Diwan. Analyzing performance traces using temporal formulas. *Software: Practice and Experience*, 44(7):777–792, 2014. [p360]

S. W. Sen, H. Ford, D. R. Musicant, M. Graham, O. S. Keyes, and B. Hecht. Barriers to the localness of volunteered geographic information. *Proceedings of the 2015 ACM Conference on Human Factors in Computing*, 2015. [p364]

H. Wickham. *httr: Tools for Working with URLs and HTTP*, 2015. URL http://CRAN.R-project.org/package=httr. R package version 1.0.0. [p360, 362]

H. Wickham and R. Francois. *readr: Read Tabular Data*. URL https://github.com/hadley/readr. R package version 0.1.1.9000. [p361]

Oliver Keyes
Rapid7
One Main Street, Penthouse
Cambridge, MA, 02142, USA
ironholds@gmail.com

Bob Rudis
Rapid7
One Main Street, Penthouse
Cambridge, MA, 02142, USA
brudis@rapid7.com

Jay Jacobs
BitSight
Boston, MA
jay@beechplane.com

Nonparametric Tests for the Interaction in Two-way Factorial Designs Using R

by Jos Feys

Abstract An increasing number of R packages include nonparametric tests for the interaction in two-way factorial designs. This paper briefly describes the different methods of testing and reports the resulting *p*-values of such tests on datasets for four types of designs: between, within, mixed, and pretest-posttest designs. Potential users are advised only to apply tests they are quite familiar with and not be guided by *p*-values for selecting packages and tests.

Introduction

In his book 'Discovering Statistics Using R' (Field et al., 2012), Andy Field remarked that, contrary to the popular assertion, there are robust methods that can be used to test for the interaction in mixed models. He was referring to the **WRS** package (early version of **WRS2** by Mair et al. (2015), based on Rand Wilcox's book (Wilcox, 2012)). At that time, this apparently was the only R package known to the authors for nonparametric (robust or distribution-free) tests for the interaction in factorial designs. The **nparLD** package by Noguchi et al. (2012), which offers a variety of such tests, was first published in September 2012. Since then, an increasing number of R packages have emerged with functions to run nonparametric tests for the interaction(s) in factorial designs.

The main purpose of this paper is to familiarize researchers and potential users, who have a fair knowledge of statistics, with R packages that include nonparametric tests (R functions for such tests) for the interaction in two-way factorial designs. I first shortly describe the different methods for such tests in R Packages (available at the time of writing) and then report the resulting *p*-values of the tests, applied on data of two-way between, within, and mixed factorial designs. The term *between* refers to a between-subjects independent factor (or variable), for which a different group of subjects (or units of observation) is used for each level of the factor. A *within*-subjects factor, on the other hand, is an independent factor that is manipulated by testing each participant at each level of the factor, also named *repeated measures*. *Mixed* designs are a combination of between and within factors. For the account of *p*-values, in R packages available nonparametric functions to test for the interaction were run on datasets for four types of two-way designs: 'between x between', 'within x within', 'between x within' or 'mixed', and a special case, '(between x) pretest-posttest' designs. The latter design is a common mixed design with only two levels of the within factor.

In the next section, I advise potential users not to rely on *p*-values and to justify why they chose the particular method of testing for each of the four types of designs. They should know what the chosen test does. In the concluding section the main advices are summarized and I close with the paradox Fagerland (2012) has pointed to.

Methods and R packages

The word *nonparametric* is used here in a general sense: to include all distribution-free methods that do not rely on the restrictive assumptions of parametric tests, particularly about normality of the outcome distribution and homogeneity of variances. There are some situations when it is clear that the outcome does not follow a normal distribution. These include situations when the outcome is an ordinal variable or a rank, when there are definite outliers or when the outcome has clear limits of detection. (Data with limits of detection require quite advanced special methods for analyzing (see e.g., LaFleur et al., 2011), which are not discussed here.) Tools to address assumption problems are: simulations, nonparametric tests, robust procedures, data transformation, and re-sampling. The word nonparametric is rather associated with rank tests, and 'robust' primarily refers to methods for dealing with outliers, but I use the term nonparametric for all situations.

- An account of simulation studies would, it seems to me, not fit into the purpose of the R Journal and therefore is not covered in this paper.
- Rank test and robust methods are the main topics of interest. The word *robust* can be interpreted literally. If a test is robust, the validity of the test result will not be affected by poorly structured data. Robust also has a more technical meaning. If the actual Type I error rate of a test is close to the proclaimed Type I error rate (e.g., .05) the test is considered robust.
- Data transformation is not covered in this article. Erceg-Hurn and Mirosevich (2008) remarked that transformations often fail to restore normality and homogeneity of variances, they do not

deal with outliers, they can reduce power, they sometimes rearrange the order of the means from what they were originally, and they make the interpretation of results difficult, as findings are based on the transformed rather than the original data. Data transformation should be replaced by more up-to-day methods.

- Re-sampling techniques such as *permutation* or *randomization* tests and *bootstrap* are only very concisely described here. Permutation tests use all possible distinct permutations of the dependent variable, holding the independent variables fixed. Unfortunately, a typical full permutation test is too time-consuming. An alternative is often called a randomization test. (Many authors use both terms interchangeably.) The underlying idea of randomization tests is to compare the results from the real data against the possible results if one repeatedly (e.g., 10,000 times) re-labels the data points, then see how extreme the results from the real data are, when compared against the array of alternative arrangements of the data. There are a number of R packages for randomization tests (e.g., **coin**, **lmPerm** and **perm**), but, to my knowledge, they do not readily include test for the interaction in two-way factorial designs. The `ezPerm` function from the **ez** package by Lawrence (2015) can be used for permutation tests with many types of factorial designs. (This package also has functions for visualization of the interaction using bootstrap: `ezBoot` and `ezPlot2`. Visualization methods are beyond the scope of this paper.)

  A bootstrap is a process in which data are re-sampled repeatedly (randomly with replacement and each time of the same size as the original data), and a statistic is calculated for each re-sampling to form an empirical distribution for that statistic. The **boot** package by Canty and Ripley (2016) provides extensive facilities for bootstrapping and related re-sampling methods. This package has a function for confidence intervals: `boot.ci`.

In my opinion, nonparametric tests not only have the obvious advantage of not requiring the assumption of normality or of homogeneity of variance, but also the benefit that they can be used with many different types of scales and that, when sample size is small, there may be no alternative to use a nonparametric test unless the population distribution is known exactly. Gibbons (1993) observed that ordinal scale data, notably Likert-type scales, are very common in social sciences and argued these should be analyzed with nonparametric tests.

Dealing with outliers

Rand Wilcox's book (Wilcox, 2012) and the corresponding R package **WRS2** offer robust methods for dealing with outliers: trimmed means, bootstrap (see brief description above), median tests and M-estimators.

Trimmed means This involves the calculation of the mean after discarding given parts of a probability distribution or sample at the high and low end, and typically discarding an equal amount of both. This number of points to be discarded is usually given as a percentage of the total number of points, but may also be given as a fixed number of points. The `t2way` and `bwtrim` functions from **WRS2** are based on 20% trimmed means, respectively for between x between and mixed (between x within) designs.

Median tests The median is a robust measure of central tendency (the mean is not), thus not influenced by outliers; therefore median tests are often chosen for dealing with outliers. The `med2way` function from **WRS2** is such a test.

M-estimators M-estimators are a general class of robust statistics which are obtained as the minima of sums of functions of the data, e.g., iterated re-weighted least-squares. As already mentioned, in the **WRS2** package, the `t2way` function computes a between x between ANOVA for trimmed means with interactions effects. The accompanying `pbad2way` performs a two-way ANOVA using M-estimators for location. With this function, the user can choose between three M-estimators for group comparisons: M-estimator of location using Huber's ψ, a modified ψ estimator, or a median. In the same package the `bwtrim` function computes a between x within (mixed) subjects ANOVA on the trimmed means. Along with this function, the `sppbi` function computes the interaction effect, using bootstrap. With this function, the user here too can choose between the same three M-estimators for group comparisons.

Ordinal data and (aligned) ranks

The vast majority of nonparametric tests are rank-based tests. Many authors have proposed their own methods of ranking to test for the interaction. A special method, the alignment of the data before ranking, was introduced early in the 1990s (see e.g., Higgins et al., 1990). Aligning implies that some estimate of a location (e.g., for the effect on a certain level of a given factor), such as the mean or median of the observations, is subtracted from each observation. These data, thus aligned according

to the desired main or interaction effect, are then ranked and parametric tests are performed on the aligned ranks. Higgins and Tashtoush (1994) offered formulas for aligning the data with completely random (between x between) designs and for repeated measures (mixed) designs.

Aligned ranks The aligned ranks tests functions `aligned.rank.transform` (from the **ART** package by Villacorta (2015)) and `art` (from the **ARTool** package by Kay and Wobbrock (2015)) can be used for between x between designs. Both functions are aligned ranks tests based on the Higgins and Tashtoush formula for completely random designs (Higgins and Tashtoush, 1994, pp. 203-204). The `art` function can also be used for within x within designs and for higher order designs. Hettmansperger also proposed a ranking method (Hettmansperger and Elmore, 2002) to test for the interaction in between x between designs which essentially corresponds to the aligned rank transform method. To my knowledge, there is no package available (yet?) implementing this method (which is quite complicated to accomplish with a simple calculator). The `npIntFactRep` function (from the **npIntFactRep** package by Feys (2015)) yields aligned ranks tests for the interaction in two-way mixed designs, based on Beasley and Zumbo (2009), and uses the Higgins and Tashtoush formula for split-plot or repeated measures designs (Higgins and Tashtoush, 1994, pp. 208) to align the data for the interaction. It lists ANOVA tables for three types of ranks: regular, Friedman, and Koch ranks.

Rank-based tests For between x between designs, the `raov` function from the **Rfit** package by Kloke and McKean (2012) is available. This package is for the rank-based analysis of linear models, a robust alternative to least squares. This `raov` test is based on reduction in dispersion for testing main effects and interaction, using an algorithm described in Hocking (1985). Gao and Alvo (2005) developed their own ranking method to test for the interaction in such designs, by comparing the sum of row ranks with the sum of column ranks. The `interaction.test` function from the **StatMethRank** package by Quinglong (2015) is an application of this method. The already mentioned **nparLD** package offers two functions for two-way designs: the `ld.f2` function for within x within and the `f1.ld.f1` function for mixed (between x within) designs. (*ld* stands for longitudinal data.) The package also offers functions for three-way designs: `f1.ld.f2` (between x within x within) and `f2.ld.f1` (between x between x within), along with functions for confidence intervals and to help researchers choose the correct function. The functions in this package are based on studies by Akritas and Brunner (see e.g., Akritas et al., 1997). The testing method defines relative treatment effects in reference to the distributions of the variables measured in the experiment. These are estimated on mean ranks. In one sense, therefore, one can think of a relative treatment effect as a generalized expectation or mean (see e.g., Shah and Madden, 2004, for an introduction to the basic concepts underlying these tests).

Resulting p-values

In this section, the resulting *p*-values are reported for various designs with concrete datasets, obtained with the appropriate R packages tests.

Between x Between

Two-way between subjects designs are dealt with first, using the 'Box-Cox' and the 'Ants-eating-lizards' data.

Box-Cox data

The **Rfit** package uses the data from Box and Cox (1964) on the survival times (10hr units) of animals in a 3 x 4 factorial experiment (n = 4 observations per cell). (The authors, Box and Cox, gave no further details about the study than that it was a biological experiment using a 3 x 4 factorial design, the factors being (a) three poisons and (b) four treatments). The distribution of the Box-Cox data is displayed in the left panel of Figure 1. The response (dependent variable) is the log survival (`logSurv`) time of the animal.

For these data, the Fligner-Killeen (median) test for the homogeneity of variances is significant (α = .05), with a p-value = .0011, as is the Shapiro-Wilk normality test, with p = .0001. (The Shapiro-Wilk test is known to be biased by sample size. With large samples, small deviations from normality yield significant results. Thus e.g., a Q–Q plot might be required for verification in addition to the test, if one really wants to address the normality issue, which is not the case here.)

As illustrated in the left panel of Figure 1, the spreading of the data in the poisons I and especially in the poisons II condition is quite larger than in the poisons III condition (which illustrates the significance of the Fligner-Killeen test), and in the B and D treatments, survival is higher than in the other two treatments.

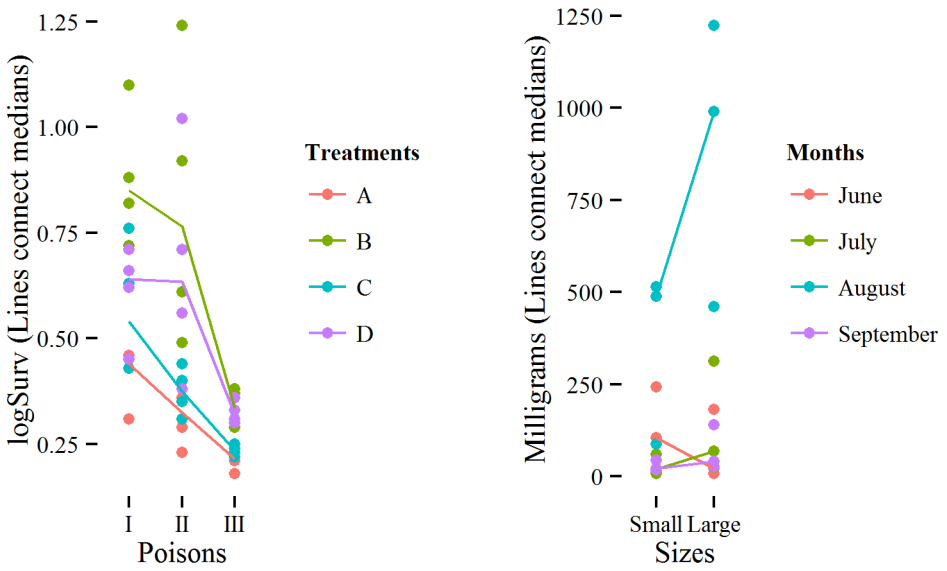


Figure 1: Distributions of the dependent variables by Between conditions for the Box-Cox (left) and Ants-eating-lizards (right) data.

In Table 1, the parametric ANOVA (*ezANOVA*, from the **ez**) on these data shows no significant interaction between treatments and poisons; neither does its permutation version *ezPerm*. The *t2way* function from **WRS2** shows no significant interaction. *pbad2way* does not run here because the covariance matrix is singular. The *med2way* function returns a significant *p*-value. The rank tests *aligned.rank.transform* (from the **ART**), *art* (from the **ARTool**) and the *raov* robust ANOVA test (from the **Rfit** package) all bring about equal and significant values, whereas the *p*-value for the ranks *interaction.test* (from the **StatMethRank**) is not significant. In the package manual, the example for the use of this function is on the Box-Cox data in matrix format.

Function	Package	Box-Cox	Ants-eating-lizards
Parametric			
ezANOVA	ez	.1123	.0617
Permutation			
ezPerm	ez	.1430	.0820
Robust			
t2way	WRS2	.0560	.3310
pbad2way	WRS2	*NA*	.5373
med2way	WRS2	.0000	.0000
Rank			
raov	Rfit	.0144	.0115
aligned.rank.transform	ART	.0168	.0688
art	ARTool	.0168	.0688
interaction.test	StatMethRank	.4913	.3959

Table 1: Resulting *p*-values of the various tests for the interaction on the Box-Cox and Ants-eating-lizards data.

Ants-eating-lizards

In the web pages material accompanying his book, in the folder on permutation tests, David Howell (Howell, 2013) illustrates the use of R scripts for permutation tests in factorial designs. He took an example from Manly (2007, p. 144). In this study, the number of ants consumed by two sizes of lizards over each of four months were observed. The distributions of milligrams of ants consumed by 24 lizards (categorized as large or small in sizes; $n = 3$ lizards per cell) during four months are displayed

in the right panel of Figure 1. It is obvious that in August, lizards – especially the large sized ones – eat most ants.

Effect	Parametric	Manly	Edgington	Still-White	ter Braak
Size	.0506	.0485	.0400	.0400	.0480
Months	.0000	.0000	.0000	.0000	.0000
S x M	.0617	.0480	.0480	.0512	.0488

Table 2: Summary of results in *p*-values from Howell (2013).

For these data, the Fligner-Killeen median test for the homogeneity of variances is not significant. Yet, the Shapiro-Wilk normality test is significant, with a *p*-value = 5.2E–06.

Howell performed several permutation tests with different approaches. The resulting *p*-values are summarized in Table 2 (as in Howell's table, *p*-values are reported only up to 4 digits after the decimal point). With the parametric test and with the Still-White approach, the *p*-values for the interaction (S x M) are 'almost' significant. (I know I should not use such terms, yet researchers typically do; this is discussed further below in the 'Choosing methods' section.) With the other approaches, the *p*-values are significant. Using the Hettmansperger method (Hettmansperger and Elmore, 2002), I calculated a very small *p*-value: 5.9E–157.

In Table 1, the resulting *p*-values for the interaction (sizes x months) with the Ants-eating-lizards data are on the right side. The parametric value using `ezANOVA` is not significant and equal to the parametric value in Table 2 (`p` = .0617). The `ezPerm` value is about the same. The `t2way` value, based on trimmed means, is not significant. The `pbad2way` function also returns a non-significant value (about .5400, depending on the ad hoc bootstrap). The `med2way` however, as for the Box-Cox data, yields a very small *p*-value. (This function only gives up to 4 digits after the decimal point.) The `raov` robust test value reveals to be significant. The aligned ranks `aligned.rank.transform` and `art` values are the same and not significant, and the *p*-value for the `interaction.test` is not significant.

Within x Within

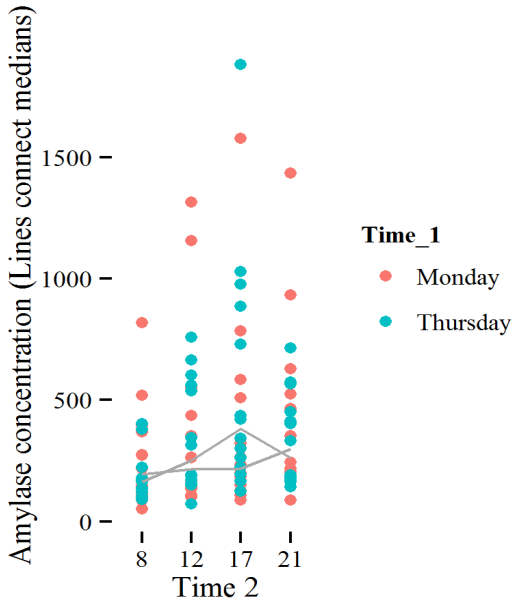


Figure 2: Distributions of the Amylase concentrations by Within conditions for the Amylase data.

Amylase data

For a two-way within (doubly repeated) subjects designs, the 'Amylase' data form the **nparLD** package were used. The data are from a longitudinal study on the concentration of α-amylase (a protein most prominent in pancreatic juice and saliva) levels (in U/ml) of the saliva from a group of 14 volunteers. Measurements were taken on 8 occasions, four times per day (8 a.m., 12 p.m., 5 p.m., 9 p.m.) and on

two days (Monday, Thursday). The distribution of the amylase concentrations in Figure 2 suggests that, on Monday, these are higher than on Thursday and that there might be an interaction between time 1 (days) and time 2 (hours). The spreading of concentrations is high at noon, and highest in the afternoon.

The Shapiro-Wilk test for normality (on the data in 'long' format) is significant with a p-value close to zero: 1.41E–12. Mauchly's sphericity test on the 8 repeated measures also reveals a significant value p = .0135. In Table 3, all p-values are significant. (The H-F value is for the Huynh-Feldt correction due

Function	Package	Amylase
Parametric		
ezANOVA	ez	.0112
– H-F corrected		.0221
Permutation		
ezPerm	ez	.0180
Rank		
art	ARTool	.0127
ld.f2	nparLD	
– Walt-type		.0025
– Anova-type		.0042

Table 3: Resulting p-values of the various tests for the interaction on the Amylase data.

to lack of sphericity.) Both values of the `ld.f2` (from **nparLD**) function are somewhat smaller than the other.

Mixed (between x within)

Three datasets were chosen for the nonparametric tests for the interaction in mixed designs: the 'Hangover', the 'Higgins', and the 'Bonate' data. The latter dataset is for a pretest-posttest mixed design, which is reviewed in the 'Pretest-Posttest' subsection.

Hangover data

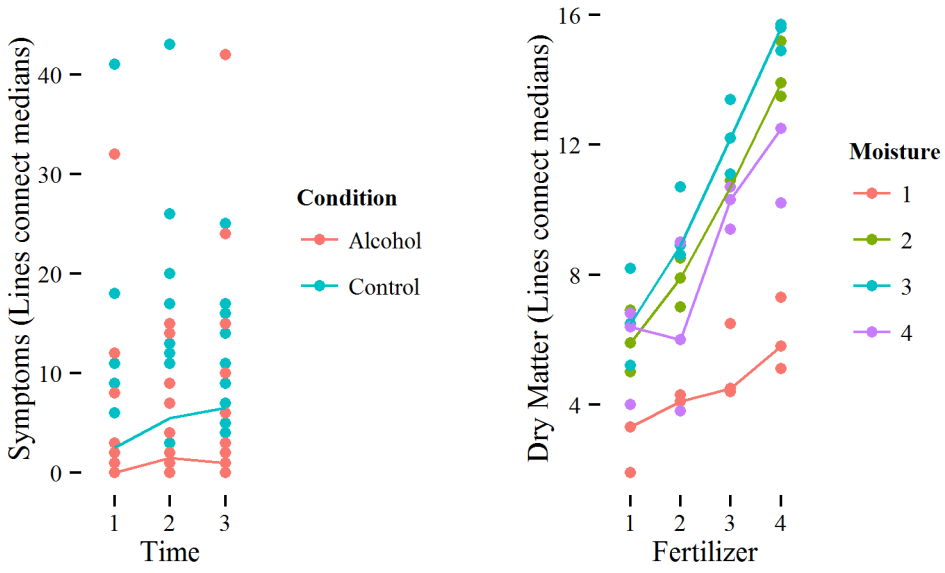


Figure 3: Distributions of the dependent variables by the Mixed conditions for the Hangover (left) and the Higgins (right) data.

The data on hangover symptoms are from Wilcox (2012, p.411). These data, also used in the **WRS2** package, come from a study on the effect of consuming alcohol, in which the number of hangover

symptoms were measured for two independent groups (N = 40, 2 x n = 20, 3 repeated measures), with each subject consuming alcohol and being measured on three different occasions. One group consisted of sons of alcoholics and the other was a control group. The distribution of this dataset is presented in the left panel of Figure 3.

The Shapiro-Wilk normality test is significant, the p-value is about zero: 4.78E–14. Mauchly's test for sphericity is not significant.

Function	Package	Hangover	Higgins
Parametric			
ezANOVA	ez	.3823	.0003
Permutation			
ezPerm	ez	.3800	.0020
Robust			
bwtrim(20%)	WRS2	.5790	*NA*
sppbi	WRS2	.8607	.0000
Rank			
npIntFactRep	npIntFactRep		
– regular		.6424	.0007
– Friedman		.7289	.0019
– Koch		.6502	.0037
art	ARTool	.3743	.0006
f1.ld.f1	nparLD		
– Walt-type		.6165	.0000
– Anova-type		.6812	.0466

Table 4: Resulting p-values of the various tests for the interaction on the Hangover and Higgins data.

None of the p-values for the interaction in the Hangover data, in the left column in Table 4, are significant.

Higgins data

The Higgins data are the Table 5 data in Higgins et al. (1990). They came from an experiment by Milliken and Johnson (1984) in which 4 peat pots, with a different (within) level of fertilizer randomly assigned to each, were placed in a tray (unit of observation). Each tray was treated with one of four different (between) moisture levels (N = 12, 4 x n = 3 trays, 4 repeated measures). The distribution of this set is displayed in the right panel of Figure 3.

Both the Shapiro-Wilk normality test and Mauchly's test for sphericity are not significant. (This implies that nonparametric tests are not really required here, but this is not an issue. Higgins et al. (1990) used these data to illustrate the aligned rank transform procedure.)

The p-values for the interaction in the Higgins data are reported in the right column of Table 4; all of them are significant. The `bwtrim` function does not run on these data because the covariance matrix is singular. `f1.ld.f1` gives the same warning; its resulting values might not be valid here.

Pretest-Posttest

This type of design is a special case of mixed designs, with only 2 within levels. Bonate (2000) thoroughly discussed the data he presented in his Table 5.4. They resulted from a study with two between groups (control and treatment; n = 10 and n = 9, respectively) and two repeated measures: pre- and posttest. In the treatment group, there was an outlier on the posttest: a value of 19 between values quite larger than 60 in the whole table. (Bonate did not give any more details about these data.)

According to Bonate, pretest-posttest data can be analyzed in several ways: (1) ANOVA on final scores alone, (2) on difference scores, (3) on percentages change scores, (4) by means of an analysis of covariance (ANCOVA) with the pre-test as covariate for the predicting group factor and the posttest as outcome variable, (5) blocking by initial scores (stratification), and (6) as repeated measures. For this design, I only review the ANCOVA, because most statisticians would agree that this should be the preferred method for analysis of pretest-posttest data (see e.g., Dimitrov and Rumrill, 2003). To test for the interaction in such a design boils down to the test for the between effect (predictor) on the posttest (criterion) after the pretest has been included in the regression model as a covariate.

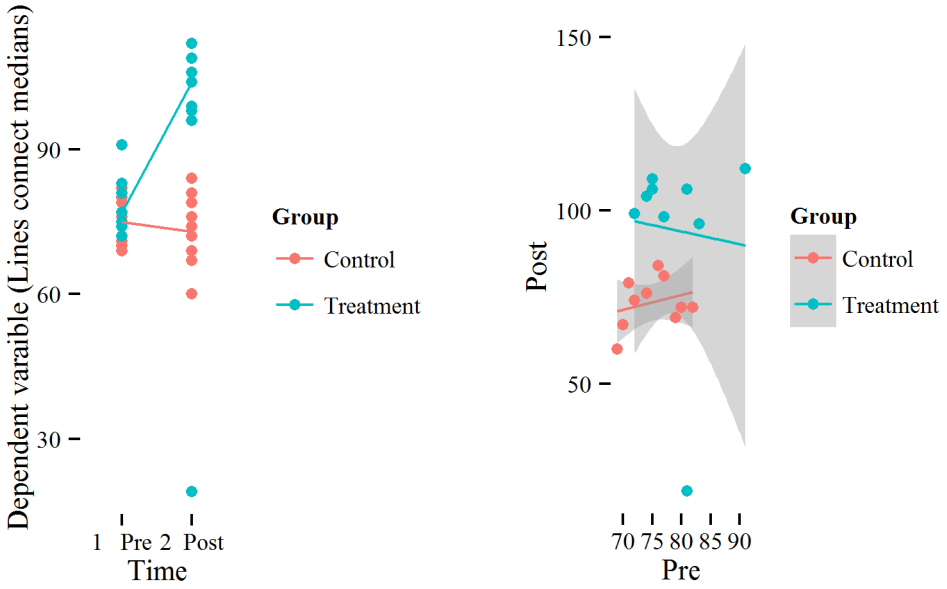


Figure 4: Distribution of the dependent variable by the Pretest-Posttest and Group conditions (left panel) and Pretest-Posttest regression plots by Group (right panel) for the Bonate data.

The Shapiro-Wilk test (on the data in 'long' format) is significant, p = .0003. Grubb's test (`grubb.test`) for one outlier (from the **outliers** package by Komsta (2011)) spots the value 19, with p = .0007.

It seems evident from Figure 4 (in the right panel) that the regression slopes are not equal between groups. (Different scales for the pre- vs. posttest were used for the plot to fit in the whole figure, despite the outlier. The shaded areas correspond to the 95% confidence intervals.) So, one of the assumptions for an ANCOVA, that of homogeneity of regression slopes, seems to be violated, most probably due to the outlier. Yet, the robust `onecova` function (from the **npsm** package by Kloke and McKean (2015), based on the their book (Kloke and McKean, 2011)), shows that the interaction group x pretest is not significant, p = .7457. Furthermore, when comparing the Pearson correlations between pre- and posttest in the two groups (r = .2683 in the control group and –.0756 in the treatment group) with the `cocor.indep.groups` function (from the **cocor** package by Diedenhofen and Musch (2015)), the resulting p-value is not significant: p = .5284. Based upon these reassuring results a nonparametric ANCOVA on these data seems justified.

Robust and rank (R)ANCOVA

Except for the outlier, in Figure 4 (left panel), all posttest values are clearly much higher in the treatment group (green dots) than in the control group (red dots). Yet, a parametric ANCOVA on these data yields a non-significant group effect (which corresponds to the interaction group x pre-posttest), with a p-value = .0576.

Bonate (2000, pp.103-106) proposed two ways for dealing with an outlier: simply removing the outlier or applying a method to minimize the influence of an observation on parameter estimations, namely the iterative re-weighted least-squares (IRWLS). He used two weight functions for the iterations: the Huber function and the bisquare. Removing the outlier from the data in this example resulted in a p-value (for the group effect) close to zero (<.0001), as did both weight functions.

Quade (1967) introduced the RANCOVA, the rank analysis of covariance. It is an ANOVA for the between effect on the residuals from the regression of the ranked posttest (criterion) on the ranked pretest (covariate). This test is quite significant here: p = .0048. A regression analysis of the ranked posttest (criterion) on ranked pretest and treatment (covariate and predictor) is another way to run a RANCOVA. It yields a comparable p-value = .0031. These values confirm Bonate's results.

The **WRS2** package offers several robust alternative for the ANCOVA: `ancov`, `ancboot`, etc. . Unfortunately, these functions fail when the number of degrees of freedom is smaller than or equal to 2, as is the case here. The `onecovahomog` function from **npsm**, which is a robust ANCOVA under homogeneous slopes, yields a p-value = .0001.

As another alternative for ANCOVA, one can run any of the many (in **WRS2** as well as in many

other packages) nonparametric or robust tests for one-way designs on variables, i.c. the residuals from the regression analysis of the posttest on the pretest. For example, the `yuen` function yields a p-value = .0001, with a trimmed mean difference between residuals of –27.24 and a 95% confidence interval: –36.10 to –18.37. The Kruskal-Wallis rank sum test returns a p-value = .0043. The Exact Wilcoxon-Mann-Whitney test yields a p-value = .0030. These tests can also be run on gain or difference scores, but many statisticians would not recommend (to say the least) analyzing such scores (see e.g., Senn, 2006).

Choosing between methods

Many statisticians (see e.g., Gardner and Altman, 1986), have warned researchers against using the accept/reject philosophy of hypothesis testing. Researchers tend to express joy on achieving a p-value of .049 and despair on finding one of 'only' .051. They seem to internalize between these values as 'right' versus 'wrong', or even of 'renewal of grants' versus 'termination of a research career'. This philosophy has led to the publication bias in journals (see e.g., Easterbrook et al., 1991), because research with statistically significant results is potentially more likely to be submitted than studies with non-significant results, with this bias of over-representation of positive results as a consequence.

The alternative, according to Gardner and Altman (1986), is to estimate the magnitude of the difference of a measured outcome between treatment groups, along with some interval that includes the population value of the difference with some specified probability: the confidence interval.

In this respect, I would advise researchers who face a choice between the increasing number of nonparametric R packages with functions for testing the interaction in factorial designs, not to compare the resulting p-values as reported in the previous section. They should certainly not go shopping for a test with the smallest p-values. Instead, they should be guided by the reason why a nonparametric test was indicated and their knowledge about how the chosen test deals with this reason why.

The above reported resulting p-values sometimes are quite different for the same datasets, depending upon the in R packages available types of tests. Since the number of available packages varies with the type of design, I discuss this choice issue for each of these separately.

Between x Between

The resulting p-values for the interaction in the between design data, summarized in Table 1, are quite divergent. With the publication strategy in mind, a researcher might very well be tempted to go for the robust median test `med2way` from **WRS2**. Yet, this test is intended to deal with outliers. If a plot of the data, e.g., a box-plot does not show any evidence of outliers, then this would not be an acceptable choice. The relevant aspect of the median test is that it only considers the position of each observation relative to the overall median, i.e., the number of times (frequency) the observation is above or below the overall median. This might not be what the researcher wants. Freidlin and Gastwirth (2000) argued the median test should be retired from general use.

For ordinal data, the researcher has a choice between the `raov`, `aligned.rank.transform`, `art` and, `interaction.test` functions; they yield comparable results. His/her choice should be inspired by the degree of knowledge he/she has or wants to invest in the rationale behind these tests. A researcher who is familiar with regression models might prefer the `raov` function; a researcher who is more acquainted with traditional ANOVA-type tests might choose one of two the aligned rank test functions. They essentially do the same thing; the `aligned.rank.transform` function has a somewhat more detailed output than `art`. The `interaction.test` yields puzzling results, because for both datasets, its p-values are quite different compared to all the other values in Table 1. Personally, I do not quite understand what this test does and therefore would not use it. Note that the permutation test `ezPerm` yields p-values not too much different from its parametric equivalent `ezANOVA`.

Within x Within

For the within x within design, all the resulting p-values in Table 3 are about equal ($p < .0200$). They all are quite significant. The two `ld.f2` type tests from **nparLD** yield the lowest values. The `art` function does not allow to specify the model for the covariance structure of the repeated measures. For doubly repeated measures designs, this might be a shortcoming. In the literature, I could not find any discussion about how to apply the aligned rank transform for doubly repeated measures. Since the `art` function uses the formula for between x between design, this might not be the right one for within x within designs. I would advise the use the `ld.f2` function here; it is better documented and was developed especially for such designs.

Mixed (between x within)

The resulting p-values in the mixed designs, displayed in Table 4, are not much dissimilar, but there are some peculiarities. For the Hangover data, the parametric test value is $p = .3823$. The nonparametric (robust and rank) p-values all are about or above .60, except for the `art` value. For the Higgins data, the anova-type value from `f1.ld.f1` seems a little odd, because it is only just under the α-level, whereas all the other values clearly are significant. As noted above, `f1.ld.f1` gave a warning; one should not choose this function here.

It is somewhat puzzling that the `art` function does not yield the same p-values as the regular aligned ranks test from **npIntFactRep**, because they supposedly both use the same procedure to align the data for the interaction, i.e. the formula for split-plot or repeated measures designs from Higgins and Tashtoush (1994, p. 208). Especially for the Hangover data the values are quite different (.6424 vs. .3743).). For the Higgins dataset, I calculated the Pearson correlation between the data aligned for the interaction with the `art` function from **ARTool** and those same data aligned for the interaction with the Higgins and Tashtoush formula for split-plot designs: this was (only) $r = .80$. When applying the Higgins and Tashtoush formula for two-way completely random designs (Higgins and Tashtoush, 1994, p. 203–204) to align the data for the interaction and then running a repeated measures ANOVA on the rank aligned data, with the *compound symmetry* options for the covariance structure (with the SAS mixed procedure described in Littell et al. (1996)), I found exactly the same value as for `art` in Table 4: $p = .3743$. This indicates that the `art` function from **ARTool** uses the wrong Higgins and Tashtoush formula for aligning the data in mixed designs. It uses the formula for between x between designs, instead of using the formula for mixed (In Higgins's terms: split-plot or repeated measures) designs. I therefore would recommend not to use the `art` function for such designs. Data with outliers should be analyzed with a robust function (or both), ordinal data can be analyzed either with `f1.ld.f1` (if it does not give a warning) or with `npIntFactRep`. Researcher's choice should be guide be his/here knowledge of these tests.

Pretest-Posttest

The parametric and permutation tests p-values for the group effect (indication the interaction) are not significant. Yet, the `onecovahomog` (robust ANCOVA) function is significant: $p = .0001$. The RANCOVAs and the nonparametric tests on residuals also all are quite significant. The p-values from the R package functions thus corroborate Bonate's results. In this context, researchers again should be guided by their knowledge about the particular tests/functions.

Conclusions

For all types of designs, the randomization (or permutation) and the parametric version of the tests yield comparable p-values. Therefore, I would not advise to use randomization tests as an genuine nonparametric alternative.

Given the sometimes quite divergent resulting p-values, potential users of nonparametric R functions to apply tests for the interaction in two-way factorials designs should be careful in their choices. They should not go shopping for the test function with the smallest p-value. Instead, a close examination and justification of the chosen function is recommended. They should know exactly what the chosen test does.

Finally, I would like to mention the paradox Fagerland (2012) has pointed to, namely that as sample sizes of research studies have increased, the use of nonparametric tests has also escalated at the expense of parametric tests. This is a paradox because parametric tests, like t-tests, are quite robust when samples sizes are large. Fagerland has shown that using nonparametric tests in large studies may provide answers to the wrong question. He stated that nonparametric tests are most useful for small-sized studies.

Bibliography

M. G. Akritas, S. F. Arnold, and E. Brunner. A unified approach to rank tests for mixed models. *Journal of Statistical Planning and Inference*, 61(2):249–277, 1997. [p369]

T. M. Beasley and B. D. Zumbo. Aligned rank tests for interactions in split-plot designs: Distributional assumptions and stochastic homogeneity. *Journal of Modern Applied Statistical Methods*, 8 (1):16–50, 2009. URL http://www.soph.uab.edu/Statgenetics/People/MBeasley/Beasley-Zumbo-AlignedRanks-JMASM-2009.pdf. [p369]

P. L. Bonate. *Analysis of Pretest-Posttest Designs*. Chapman-Hall, 2000. [p373, 374]

G. Box and D. Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, 26(2):211–252, 1964. URL http://fisher.osu.edu/~schroeder.9/AMIS900/Box1964.pdf. [p369]

A. Canty and B. D. Ripley. *boot: Bootstrap R (S-Plus) Functions*, 2016. R package version 1.3-18. [p368]

B. Diedenhofen and J. Musch. cocor: A comprehensive solution for the statistical comparison of correlations. *PloS ONE*, 10(4):1–12, 2015. URL http://dx.doi.org/10.1371/journal.pone.0121945. [p374]

D. M. Dimitrov and P. D. Rumrill. Pretest-posttest designs and measurement of change. *Work*, 20(2): 159–165, 2003. URL http://www.ncbi.nlm.nih.gov/pubmed/12671209. [p373]

P. J. Easterbrook, J. A. Berlin, R. Gopalan, and D. R. Matthews. Publication bias in clinical research. *Lancet*, 337(8746):867–872, 1991. [p375]

D. M. Erceg-Hurn and V. M. Mirosevich. Modern robust statistical methods: an easy way to maximize the accuracy and power of your research. *American Psychologist*, 63(7):591–601, 2008. [p367]

M. W. Fagerland. t-tests, non-parametric tests, and large studies – a paradox of statistical practice? *BMC Medical Research Methodology*, 12:78, 2012. URL http://bmcmedresmethodol.biomedcentral.com/articles/10.1186/1471-2288-12-78. [p367, 376]

J. Feys. *npIntFactRep: Nonparametric Interaction Tests for Factorial Designs with Repeated Measures*, 2015. URL https://cran.r-project.org/web/packages/npIntFactRep. R package version 1.5. [p369]

A. Field, J. Miles, and Z. Field. *Discovering Statistics Using R*. SAGE, 2012. [p367]

B. Freidlin and J. L. Gastwirth. Should the median test be retired from general use? *The American Statistician*, 54(3):161–164, 2000. URL http://www.jstor.org/stable/2685584. [p375]

X. Gao and M. Alvo. A nonparametric test for interaction in two-way layout. *The Canadian Journal of Statistics*, 33(4):529–543, 2005. URL http://onlinelibrary.wiley.com/doi/10.1002/cjs.5550330405/pdf. [p369]

M. J. Gardner and D. G. Altman. Confidence intervals rather than *p*-values: Estimation rather than hypothesis testing. *British Medical Journal*, 292:746–750, 1986. [p375]

J. D. Gibbons. *Nonparametric Statistics: An Introduction*. SAGE, 1993. [p368]

T. P. Hettmansperger and R. Elmore. Tests for interaction in a two-way layout: Should they be included in a nonparametrics course? In ICOTS, editor, *Conference Proceedings*, volume 6, Cape Town, South Africa, 2002. International Association for Statistical Education. [p369, 371]

J. J. Higgins and S. Tashtoush. An aligned rank transform test for interaction. *Nonlinear World*, 1(2): 201–211, 1994. [p369, 376]

J. J. Higgins, R. C. Blair, and S. Tashtoush. The aligned rank transform procedure. In *Proceedings of the 1990 Kansas State University Conference on Applied Statistics in Agriculture*, pages 185–195, Manhattan, Kansas, 1990. Kansas State University. URL http://newprairiepress.org/agstatconference/1990/proceedings/18. [p368, 373]

R. R. Hocking. *The Analysis of Linear Models*. Brooks/Cole, 1985. [p369]

D. C. Howell. *Statistical Methods for Psychology*. Wadsworth, 8th edition, 2013. URL https://www.uvm.edu/~dhowell/. [p370, 371]

M. Kay and J. O. Wobbrock. *ARTool: Aligned Rank Transform for Nonparametric Factorial ANOVAs*, 2015. URL https://cran.r-project.org/web/packages/ARTool. R package version 0.9.5. [p369]

J. D. Kloke and J. W. McKean. *Nonparametric Statistical Methods using R*. Chapman-Hall, 2011. [p374]

J. D. Kloke and J. W. McKean. Rfit: Rank-based estimation for linear models. *The R Journal*, 4(2):57–64, 2012. URL https://journal.r-project.org/archive/2012-2/RJournal_2012-2_Kloke+McKean.pdf. [p369]

J. D. Kloke and J. W. McKean. *npsm: Package for Nonparametric Statistical Methods using R*, 2015. URL https://cran.r-project.org/web/packages/npsm. R package version 0.5. [p374]

L. Komsta. *outliers: Tests for outliers*, 2011. URL https://cran.r-project.org/web/packages/outliers. R package version 0.14. [p374]

B. LaFleur, W. Lee, D. Billhiemer, C. Lockhart, J. Liu, and N. Merchant. Statistical methods for assays with limits of detection: Serum bile acid as a differentiator between patients with normal colons, adenomas, and colorectal cancer. *Journal of Carcinogenesis*, 10(12), 2011. [p367]

M. A. Lawrence. *ez: Easy Analysis and Visualization of Factorial Experiments*, 2015. URL https://cran.r-project.org/web/packages/ez. R package version 4.3. [p368]

R. C. Littell, G. A. Milliken, W. W. Stroup, and R. D. Wolfinger. *SAS® System for Mixed Models*. SAS Institute, Cary, NC, 1996. [p376]

P. Mair, R. Wilcox, and F. Schoenbrodt. *WRS2: A Collection of Robust Statistical Methods*, 2015. URL https://cran.r-project.org/web/packages/WRS2. R package version 0.4-0. [p367]

B. F. Manly. *Randomization, Bootstrap, and Monte Carlo Methods in Biology*. Chapman-Hall, 3rd edition, 2007. [p370]

G. A. Milliken and D. E. Johnson. *Analysis of Messy Data Vol I: Designed Experiments*. Van Nostrand Reinhold Company, 1984. [p373]

K. Noguchi, M. Latif, K. Thangavelu, F. Konietschke, Y. R. Gel, and E. Brunner. *nparLD: Nonparametric Analysis of Longitudinal Data in Factorial Experiments*, 2012. URL https://cran.r-project.org/web/packages/nparLD. R package version 2.1. [p367]

D. Quade. Rank analysis of covariance. *Journal of the American Statistical Association*, 62(320):1187–1200, 1967. [p374]

L. Quinglong. *StatMethRank: Statistical Methods for Ranking Data*, 2015. URL https://cran.r-project.org/web/packages/StatMethRank. R package version 1.3. [p369]

S. Senn. Change from baseline and analysis of covariance revisited. *Statistics in Medicine*, 25(24): 4334–4344, 2006. URL http://www.ncbi.nlm.nih.gov/pubmed/16921578. [p375]

D. A. Shah and L. V. Madden. Nonparametric analysis of ordinal data in designed factorial experiments. *Phytopathology*, 94(1):33–43, 2004. URL http://apsjournals.apsnet.org/doi/pdf/10.1094/PHYTO.2004.94.1.33. [p369]

P. J. Villacorta. *ART: Aligned Rank Transform for Nonparametric Factorial Analysis*, 2015. URL https://cran.r-project.org/web/packages/ART. R package version 1.0. [p369]

R. R. Wilcox. *Introduction to Robust Estimation and Hypothesis Testing*. Elsevier, 3rd edition, 2012. [p367, 368, 372]

Jos Feys
Faculty of Kinesiology and Rehabilitation Sciences, KU Leuven
Tervuursevest 101 - box 1500
3001 Leuven, Belgium
jos.feys@kuleuven.be

GMDH: An R Package for Short Term Forecasting via GMDH-Type Neural Network Algorithms

by Osman Dag and Ceylan Yozgatligil

Abstract Group Method of Data Handling (GMDH)-type neural network algorithms are the heuristic self organization method for the modelling of complex systems. GMDH algorithms are utilized for a variety of purposes, examples include identification of physical laws, the extrapolation of physical fields, pattern recognition, clustering, the approximation of multidimensional processes, forecasting without models, etc. In this study, the R package **GMDH** is presented to make short term forecasting through GMDH-type neural network algorithms. The **GMDH** package has options to use different transfer functions (sigmoid, radial basis, polynomial, and tangent functions) simultaneously or separately. Data on cancer death rate of Pennsylvania from 1930 to 2000 are used to illustrate the features of the **GMDH** package. The results based on ARIMA models and exponential smoothing methods are included for comparison.

Introduction

Time series data are ordered successive observations which are measured in equally or unequally spaced time. Time series data may include dependency among successive observations. Hence, the order of the data is important. Time series data appear in various areas and disciplines such as medical studies, economics, the energy industry, agriculture, meteorology, and so on. Modelling time series data utilizes the history of the data and makes forecasting using this history. At times, statistical models are not sufficient to solve some problems. Examples include pattern recognition, forecasting, identification, etc. Extracting the information from the measurements has advantages while modelling complex systems when there is not enough prior information and/or no theory is defined to model the complex systems. Selecting a model automatically is a powerful way for the researchers who are interested in the result and do not have sufficient statistical knowledge and sufficient time (Mueller et al., 1998) for an analysis.

The objective of this study is to develop an R package for forecasting of time series data. Some of recent softwares developed for time series are **glarma**, **ftsa**, **MARSS**, **ensembleBMA**, **ProbForecastGOP**, and **forecast** (Dunsmuir and Scott, 2015; Shang, 2013; Holmes et al., 2012; Fraley et al., 2011; Hyndman and Khandakar, 2008). In this study, we focused on the development of an R package for short term forecasting via Group Method of Data Handling (GMDH) algorithms. The history of GMDH-type neural network is based on works from the end of the 1960s and the beginning of the 1970s. First, Ivakhnenko (1966) introduced a polynomial, which is the basic algorithm of GMDH, to construct higher order polynomials. Also, Ivakhnenko (1970) introduced heuristic self-organization methods which constructed the main working system of GMDH algorithm. Heuristic self-organization method defines the way that the algorithm evolves, following rules such as external criteria. The GMDH method, convenient for complex and unstructured systems, has benefits over high order regression (Farlow, 1981).

Kondo (1998) proposed GMDH-type neural network in which the algorithm works according to the heuristic self-organization method. Kondo and Ueno (2006a,b) proposed a GMDH algorithm which has a feedback loop. According to this algorithm, the output obtained from the last layer is set as a new input variable, provided a threshold is not satisfied in the previous layer. The system of the algorithm is organized by a heuristic self-organization method where a sigmoid transfer function is integrated. Kondo and Ueno (2007) proposed a logistic GMDH-type neural network. The difference from a conventional GMDH algorithm was that the new one would take linear functions of all inputs at the last layer. Kondo and Ueno (2012) included three transfer functions (sigmoid, radial basis and polynomial functions) in the feedback GMDH algorithm. Srinivasan (2008) used a GMDH-type neural network and traditional time series models to forecast predicted energy demand. It was shown that a GMDH-type neural network was superior in forecasting energy demand compared to traditional time series models with respect to mean absolute percentage error (MAPE). In another study, Xu et al. (2012) applied a GMDH algorithm and ARIMA models to forecast the daily power load. According to their results, GMDH-based results were superior to the results of ARIMA models in terms of MAPE for forecasting performance.

There are some difficulties when applying a GMDH-type neural network. For example, there is no freely available software for researchers implementing the GMDH algorithms in the literature. We

present the R package **GMDH** to make short term forecasting through GMDH-type neural network algorithms. The package includes two types of GMDH structures; namely, GMDH structure and revised GMDH (RGMDH) structure. Also, it includes a variety of options to use different transfer functions (sigmoid, radial basis, polynomial, and tangent functions) simultaneously or separately. Data on the cancer death rate of Pennsylvania from 1930 to 2000 are used to illustrate the implementation of GMDH package. We compare the results to those based on ARIMA models and exponential smoothing (ES) methods.

Methodology

In this section, data preparation, two types of GMDH-type neural network structures, and estimation of a regularization parameter in regularized least square estimation (RLSE) are given.

Data preparation

Data preparation has an important role in GMDH-type neural network algorithms. To get rid of very big numbers in calculations and to be able to use all transfer functions in the algorithm, it is necessary for range of the data to be in the interval of (0, 1). If α_t is the actual time series dataset at hand, this necessity is guaranteed by the following transformation,

$$w_t = \frac{\alpha_t + \delta_1}{\delta_2} \tag{1}$$

with

$$\delta_1 = \begin{cases} |\alpha_t| + 1 & \text{if } \min(\alpha_t) \leq 0 \\ 0 & \text{if } \min(\alpha_t) > 0 \end{cases}$$

and

$$\delta_2 = \max(\alpha_t + \delta_1) + 1.$$

During the estimation and forecasting process in GMDH-type neural network algorithms, all calculations are done using the scaled data set, w_t. After all processes are ended–i.e, all predictions and forecasts are obtained–we apply the inverse transformation as follows,

$$\widehat{\alpha}_t = \widehat{w}_t \times \delta_2 - \delta_1. \tag{2}$$

Let's assume a time series dataset for t time points, and p inputs. An illustration of time series data structure in GMDH algorithms is presented in Table 1. Since we construct the model for the data with time lags, the number of observations, presented under the subject column in the table, is equal to $t - p$; and the number of inputs, lagged time series, is p. In this table, the variable called z is put in the models as a response variable, and the rest of the variables are taken into models as lagged time series x_i, where $i = 1, 2, ..., p$. The notations in Table 1 are followed throughout this paper.

Table 1: An illustration of time series data structure in GMDH algorithms

Subject	z	x_1	x_2	...	x_p
1	w_t	w_{t-1}	w_{t-2}	...	w_{t-p}
2	w_{t-1}	w_{t-2}	w_{t-3}	...	w_{t-p-1}
3	w_{t-2}	w_{t-3}	w_{t-4}	...	w_{t-p-2}
⋮	⋮	⋮	⋮	⋱	⋮
$t-p$	w_{p+1}	w_p	w_{p-1}	...	w_1

A better model which explains the relation between response and lagged time series is captured via transfer functions. The sigmoid, radial basis, polynomial, and tangent functions, presented in Table 2, are mainly used to explain the relation between inputs and output in GMDH-type neural network algorithms (Kondo and Ueno, 2012). We use all transfer functions, stated in Table 2, simultaneously in each neuron. In other words, we construct four models at each neuron, and then the model which gives the smallest prediction mean square error (PMSE) is selected as the current transfer function at the corresponding neuron.

Table 2: Transfer functions

Sigmoid Function	$z = 1/(1 + e^{-y})$
Radial Basis Function	$z = e^{-y^2}$
Polynomial Function	$z = y$
Tangent Function	$z = \tan(y)$

GMDH algorithm

GMDH-type neural network algorithms are modeling techniques which learn the relations among the variables. In the perspective of time series, the algorithm learns the relationship among the lags. After learning the relations, it automatically selects the way to follow in algorithm. First, GMDH was used by Ivakhnenko (1966) to construct a high order polynomial. The following equation is known as the Ivakhnenko polynomial given by

$$y = a + \sum_{i=1}^{m} b_i \cdot x_i + \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij} \cdot x_i \cdot x_j + \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{m} d_{ijk} \cdot x_i \cdot x_j \cdot x_k + \ldots \tag{3}$$

where m is the number of variables and $a, b, c, d, \ldots$ are coeffients of variables in the polynomial, also named as weights. Here, y is a response variable, x_i and x_j are the lagged time series to be regressed. In general, the terms are used in calculation up to square terms as presented below,

$$y = a + \sum_{i=1}^{m} b_i \cdot x_i + \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij} \cdot x_i \cdot x_j \tag{4}$$

The GMDH algorithm considers all pairwise combinations of p lagged time series. Therefore, each combination enters each neuron. Using these two inputs, a model is constructed to estimate the desired output. In other words, two input variables go in a neuron, one result goes out as an output. The structure of the model is specified by Ivakhnenko polynomial in equation 4 where $m = 2$. This specification requires that six coefficients in each model are to be estimated.

The GMDH algorithm is a system of layers in which there exist neurons. The number of neurons in a layer is defined by the number of input variables. To illustrate, assume that the number of input variables is equal to p, since we include all pairwise combinations of input variables, the number of neurons is equal to $h = \binom{p}{2}$. The architecture of GMDH algorithm is illustrated in Figure 1 when there are three layers and four inputs.

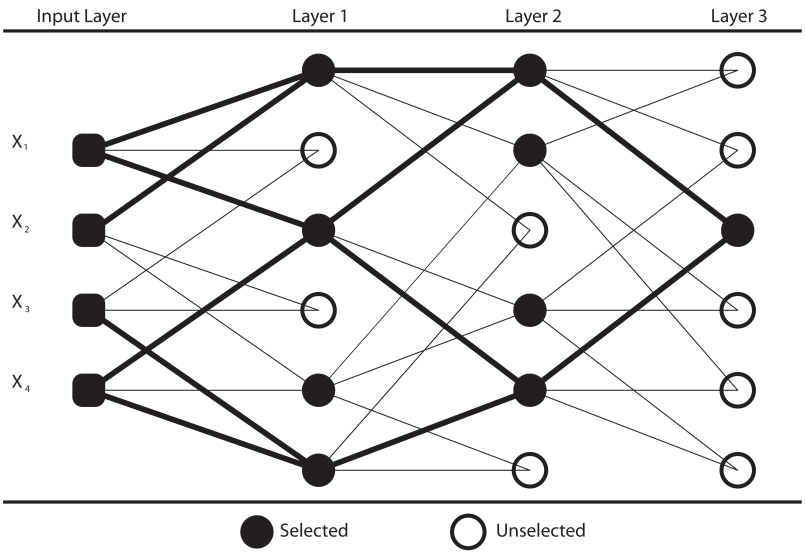


Figure 1: Architecture of GMDH algorithm

In the GMDH architecture shown in Figure 1, since the number of inputs is equal to four, the number of nodes in a layer is determined to be six. This is just a starting layer to the algorithm. The coefficients of equation 4 are estimated in each neuron. By using the estimated coefficients and input variables in each neuron, the desired output is predicted. According to a chosen external criteria, p

neurons are selected and $h - p$ neurons are eliminated from the network. In this study, prediction mean square error (PMSE) is used as the external criteria. In Figure 1, four neurons are selected while two neurons are eliminated from the network. The outputs obtained from selected neurons become the inputs for the next layer. This process continues until the last layer. At the last layer, only one neuron is selected. The obtained output from the last layer is the predicted value for the time series at hand. The flowchart of the algorithm is depicted in Figure 2.

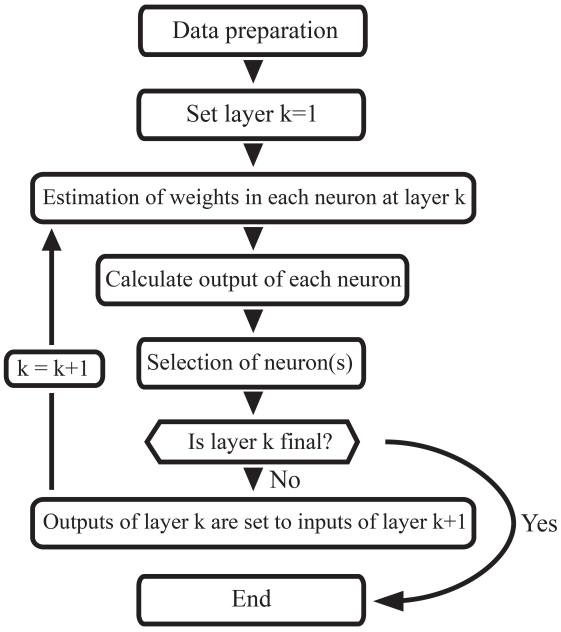


Figure 2: Flowchart of GMDH algorithms

In a GMDH algorithm, there exist six coefficients to be estimated in each model. Coefficients are estimated via RLSE.

RGMDH algorithm

A GMDH-type neural network constructs the algorithm by investigating the relation between two inputs and the desired output. Architecture of a revised GMDH (RGMDH)-type neural network does not only consider this relation, but it also considers the individual effects on the desired output (Kondo and Ueno, 2006b). There are two different types of neurons in an RGMDH-type neural network. In the first type of neuron, it is same as in GMDH-type neural network, given as in equation 4. That is, two inputs enter the neuron, one output goes out. In the second type of neuron, r inputs enter the neuron, one output goes out. This second type neuron is given by

$$y = a + \sum_{i=1}^{r} b_i \cdot x_i \quad , \quad r \leq p, \tag{5}$$

where r is the number of inputs in the corresponding second type neuron.

As mentioned above, there exist $h = \binom{p}{2}$ neurons in one layer in a GMDH-type neural network. In addition to this, with the p neurons from the second type of neuron, the number of neurons in one layer becomes $\eta = \binom{p}{2} + p$ in an RGMDH-type algorithm. The architecture of an RGMDH-type algorithm is shown in Figure 3 for the case when there are three layers and three inputs. In this architecture, since the number of inputs is three, the number of nodes in a layer is determined to be six. Here, three of six nodes are the first type of neurons in which all pairwise combinations of lagged time series are already used as in the GMDH algorithm. The rest of the three nodes are the second type of neurons where the individual effects of the lagged time series are sequentially added to the layer starting from lag 1. In each neuron, coefficients of models are calculated by using the corresponding models in equations 4 and 5. For instance, in Figure 3, there are six coefficients to be estimated as given by equation 4 for the first type of neurons, and two, three and four coefficients are estimated as given in equation 5 for the the second type of neurons when r equals to 1, 2 and 3, respectively. The desired output is predicted by utilizing estimated coefficients and input variables in each neuron. Here, p neurons are selected as living cells and $\eta - p$ death cells are eliminated from the network according to the external criteria. The rest of the algorithm is same with GMDH.

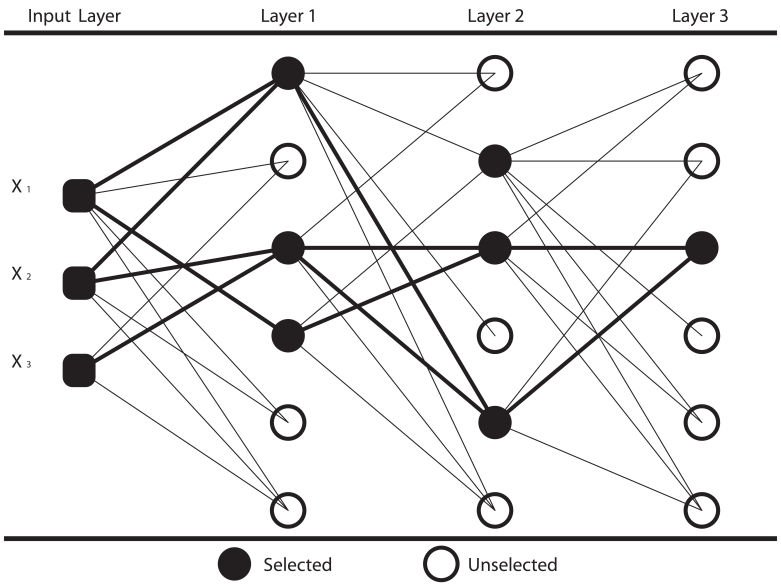


Figure 3: Architecture of an RGMDH algorithm

Estimation of regularization parameter in RLSE

In each estimation step, there exist the coefficients to be estimated. While we are estimating these coefficients, we use the regularized least square estimation method. It is stated that regularized least square estimation is utilized when there is the possibility of a multi-collinearity problem by integrating a regularization parameter, λ, into the estimation step. It is important to note that regularized least square estimation differs from the least square estimation when the regularization parameter is not zero.

We integrate the estimation of a regularization parameter (penalizing term) via validation in GMDH algorithms. For this purpose, we divide the data into two parts: a learning set and a testing set. In the **GMDH** package, 70% of the time series, by default, is taken for the learning set. Since the data set is time dependent, the order of data is saved in this division process. In other words, by default, the first 70% of the data is used for learning set and the last 30% of the data is utilized as a testing set. This whole process is applied for each model constructed in each neuron. The algorithm for the regularization parameter estimation is as follows:

i) Clarify the possible regularization parameter, $\lambda = 0, 0.01, 0.02, 0.04, 0.08, \ldots, 10.24$. Note that, when $\lambda = 0$, RLSE is converted to LSE.

ii) For each possible λ value, coefficients are estimated via RLSE by using the learning set.

iii) After the calculation of coefficients, calculate the predicted values by utilizing the test set to obtain the MSE for each regularization parameter.

iv) Select the regularization parameter which gives the minimum MSE value.

Implementation of GMDH package

The data used in this application of the **GMDH** package are the yearly cancer death rate (per 100,000 population) in the Pennsylvania between 1930 and 2000. The data were documented in Pennsylvania Vital Statistics Annual Report by the Pennsylvania Department of Health in 2000 (Wei, 2006). This dataset is also available as a dataset in the package **GMDH**. After installing the **GMDH** package, it can be loaded into an R workspace by

```
R> library("GMDH")
R> data("cancer")  # load cancer data
```

After the cancer death rate data set is loaded, one may use `fcast` function in **GMDH** package for short-term forecasting. To utilize the GMDH structure for forecasting, method is set to "GMDH". One should set the method to "RGMDH" to use the RGMDH structure.

```
R> out = fcast(cancer[1:66], method = "GMDH", input = 15, layer = 1, f.number = 5,
level = 95, tf = "all", weight = 0.70, lambda = c(0, 0.01, 0.02, 0.04, 0.08, 0.16,
0.32, 0.64, 1.28, 2.56, 5.12, 10.24))

   Point Forecast    Lo 95    Hi 95
67       249.5317 244.9798 254.0836
68       249.6316 244.4891 254.7741
69       248.9278 243.0318 254.8239
70       247.0385 240.7038 253.3731
71       244.7211 237.1255 252.3168

# display fitted values
R> out$fitted

# return residuals
R> out$residuals

# show forecasts
R> out$mean
```

In this part, we divided the data into two parts for the aim of observing the ability of methods on prediction ($n = 66$) and forecasting ($n = 5$). We include ARIMA models and ES methods for comparison purpose. For the determination of the best order of ARIMA models and the best method of ES techniques, there are two functions in the R package **forecast**. These functions, `auto.arima` and `ets`, which use grid search, select the best model according to the criteria of either AIC, AICc or BIC. For this data set, the functions suggested the model ARIMA (1, 1, 0) with intercept and an ES method with multiplicative errors, additive damped trend and no seasonality (M, Ad, N), respectively. We also added the model ARIMA (0, 1, 0) with intercept for this data set suggested by Wei (2006). For all models, prediction mean square error (PMSE) and forecasting mean square error (FMSE) are stated in Table 3.

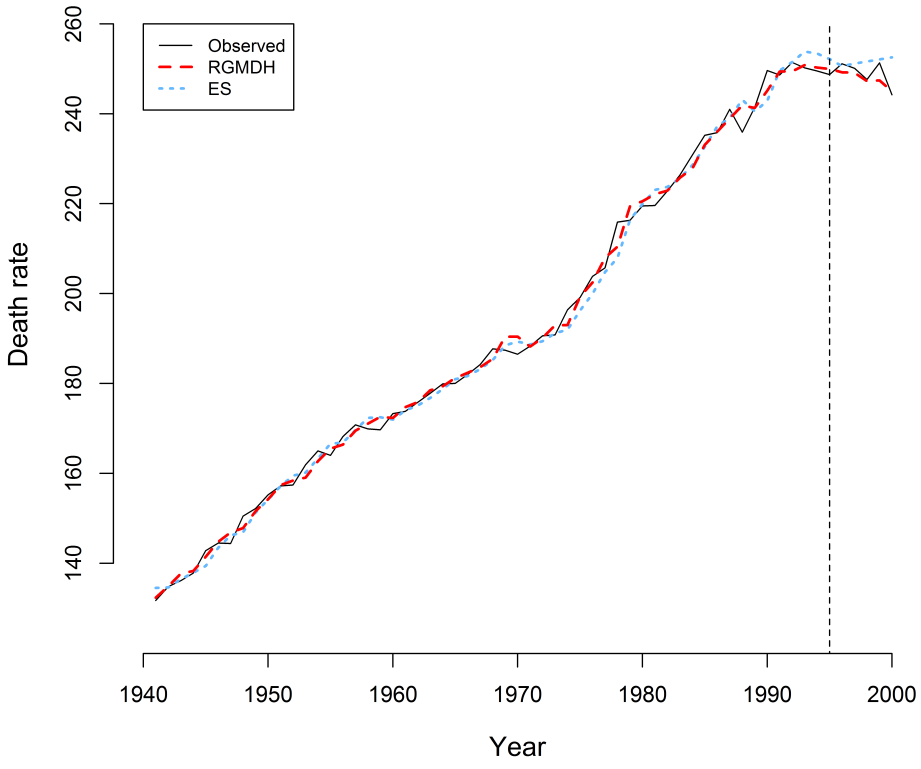


Figure 4: Yearly cancer death rate (per 100,000 population) in Pennsylvania between 1941 and 2000 with predictions and forecasts obtained via RGMDH and ES(M,Ad,N)

Table 3: Comparison of the GMDH algorithms with other models on cancer death rate in terms of prediction mean square error (PMSE) and forecasting mean square error (FMSE)

	PMSE	FMSE
GMDH	4.985	4.575
RGMDH	4.287	4.102
ARIMA(1, 1, 0) with intercept	5.995	81.874
ARIMA(0, 1, 0) with intercept	6.324	73.756
ES (M, Ad, N)	6.153	17.508

The best forecasting performance belongs to the RGMDH algorithm and its prediction accuracy also yields better results as compared to the GMDH, ARIMA and ES models. Moreover, the GMDH algorithm outperforms the ARIMA and ES models in prediction and forecasting. To avoid visual pollution in Figure 4, we include only the predictions and forecasts of RGMDH algorithm and ES (M, Ad, N).

Conclusion

In this study, we used GMDH-type neural network algorithms, the heuristic self-organization method for the modelling of complex systems, to make forecasts for time series data sets. Our primary focus was to develop a free software implementation. Concretely, we developed an R package **GMDH** to make forecasting in the short term via GMDH-type neural network algorithms. Also, we included different transfer functions (sigmoid, radial basis, polynomial, and tangent functions) into the **GMDH** package. Our R package allows that these functions can be used simultaneously or separately, as desired.

In the estimation of coefficients, since we construct the model for the data with lags, there exists a high possibility of there occurring a multi-collinearity problem. Therefore, we utilized regularized least square estimation to handle such occurences. It is important to note that estimation of a regularization parameter is the question of interest. Validation was applied in order to estimate the regularization term. After selection of a regularization term, coefficients were estimated by the help of all observations and the regularization parameter.

Application of the algorithms on a real life dataset suggests improved performance of GMDH-type neural network algorithms over ARIMA and ES models in prediction and short term forecasting. Researchers are able to use GMDH algorithms easily since our R package **GMDH** is available on Comprehensive R Archive Network (CRAN) at http://CRAN.R-project.org/package=GMDH.

Future studies are planned in the direction of transfer functions. In this study, we used four different transfer functions - sigmoid, radial basis, polynomial, and tangent functions - into GMDH algorithms. We plan to integrate the Box-Cox transformation into GMDH algorithms. GMDH algorithms with four transfer functions and GMDH algorithms with Box-Cox transformation are going to be performed on real data applications to compare the prediction and short term forecasting. After being documented, the related GMDH algorithms with the Box-Cox transformation are going to be implemented in the R package **GMDH**.

Acknowledgment

We thank the anonymous reviewers for their constructive comments and suggestions which helped us to improve the quality of our paper.

Bibliography

W. T. M. Dunsmuir and D. J. Scott. The glarma package for observation-driven time series regression of counts. *Journal of Statistical Software*, 67(7):1–36, 2015. doi: 10.18637/jss.v067.i07. [p379]

S. J. Farlow. The GMDH algorithm of Ivakhnenko. *The American Statistician*, 35(4):210–215, 1981. [p379]

C. Fraley, A. E. Raftery, T. Gneiting, J. Sloughter, and V. J. Berrocal. Probabilistic weather forecasting in R. *The R Journal*, 3(1):55–63, 2011. [p379]

E. E. Holmes, E. J. Ward, and K. Wills. MARSS: Multivariate autoregressive state-space models for analyzing time-series data. *The R Journal*, 4(1):30, 2012. [p379]

R. Hyndman and Y. Khandakar. Automatic time series forecasting: The forecast package for R. *Journal of Statistical Software*, 27(3):1–22, 2008. [p379]

A. Ivakhnenko. The group method of data handling–a rival of the method of stochastic approximation. *Soviet Automatic Control*, 13(3):43–55, 1966. [p379, 381]

A. Ivakhnenko. Heuristic self-organization in problems of engineering cybernetics. *Automatica*, 6(2): 207–219, 1970. [p379]

T. Kondo. GMDH neural network algorithm using the heuristic self-organization method and its application to the pattern identification problem. In *SICE'98. Proceedings of the 37th SICE Annual Conference. International Session Papers*, pages 1143–1148. IEEE, 1998. [p379]

T. Kondo and J. Ueno. Medical image recognition of the brain by revised GMDH-type neural network algorithm with a feedback loop. *International Journal of Innovative Computing, Information and Control*, 2(5):1039–1052, 2006a. [p379]

T. Kondo and J. Ueno. Revised gmdh-type neural network algorithm with a feedback loop identifying sigmoid function neural network. *International Journal of Innovative Computing, Information and Control*, 2(5):985–996, 2006b. [p379, 382]

T. Kondo and J. Ueno. Logistic GMDH-type neural network and its application to identification of X-ray film characteristic curve. *JACIII*, 11(3):312–318, 2007. [p379]

T. Kondo and J. Ueno. Feedback GMDH-type neural network and its application to medical image analysis of liver cancer. In *42th ISCIE international symposium on stochastic systems theory and its applications*, pages 81–82, 2012. [p379, 380]

J. A. Mueller, A. Ivachnenko, and F. Lemke. GMDH algorithms for complex systems modelling. *Mathematical and Computer Modelling of Dynamical Systems*, 4(4):275–316, 1998. [p379]

H. L. Shang. ftsa: An R package for analyzing functional time series. *The R Journal*, 5(1):64–72, 2013. URL http://journal.r-project.org/archive/2013-1/shang.pdf. [p379]

D. Srinivasan. Energy demand prediction using GMDH networks. *Neurocomputing*, 72(1):625–629, 2008. [p379]

W. W. S. Wei. *Time series analysis: univariate and multivariate methods*. Addison-Wesley publ, 2006. [p383, 384]

H. Xu, Y. Dong, J. Wu, and W. Zhao. Application of GMDH to short-term load forecasting. In *Advances in Intelligent Systems*, pages 27–32. Springer-Verlag, 2012. [p379]

Osman Dag
Department of Biostatistics
Faculty of Medicine
Hacettepe University
06100 Ankara, Turkey
osman.dag@hacettepe.edu.tr

Ceylan Yozgatligil
Department of Statistics
Faculty of Arts and Sciences
Middle East Technical University
06531 Ankara, Turkey
ceylan@metu.edu.tr

sbtools: A Package Connecting R to Cloud-based Data for Collaborative Online Research

by Luke A Winslow, Scott Chamberlain, Alison P Appling and Jordan S Read

Abstract The adoption of high-quality tools for collaboration and reproducibile research such as R and Github is becoming more common in many research fields. While Github and other version management systems are excellent resources, they were originally designed to handle code and scale poorly to large text-based or binary datasets. A number of scientific data repositories are coming online and are often focused on dataset archival and publication. To handle collaborative workflows using large scientific datasets, there is increasing need to connect cloud-based online data storage to R. In this article, we describe how the new R package **sbtools** enables direct access to the advanced online data functionality provided by ScienceBase, the U.S. Geological Survey's online scientific data storage platform.

Introduction

Cloud data storage platforms can be a powerful tool for research collaboration, distributed computing, and data publication. However, while browser-based graphical interfaces make these platforms accessible to a wide audience, there is also a need for scripted data access and manipulation so that researchers can capture data provenance and create reproducible analyses. For example, Figshare (Singh, 2011), DataOne (Michener et al., 2012), Dataverse (King, 2007) and CKAN (Winn, 2013) are all research data sharing platforms that are gaining use in different research fields. For each of these platforms, the community has released (Boettiger et al.; Leeper, 2013; Chamberlain) or is developing (Jones et al., 2013) R packages to streamline the storage and access to archived data.

These existing projects primarily focus on creating more useful, open, and accessible end products of research, but data- and code-intensive collaborative projects increasingly need collaborative solutions for data storage, sharing, and updating not just at the end of the project, but throughout the full project lifecycle. Github is increasingly used to collaborate around data products (Gandrud, 2013b), but does not scale well to the distribution of large datasets (Delcambre, 2013) and does not include metadata or queries beyond free-text search. Furthermore, most data archive platforms like DataOne and Dataverse focus on academic research projects and do not support some of the unique needs of federal research. For example, many government institutions require an archived copy of released data to be stored and available through federally operated websites. Third party storage providers are insufficient.

To address future scientific data sharing and archival challenges, the U.S. Geological Survey (USGS) created ScienceBase (`https://www.sciencebase.gov/`). This platform is designed to support the full project data lifecycle and has seen rapid adoption with USGS researchers and collaborators. ScienceBase supports the storage and access of large files and datasets. It allows data to be stored with a user-configurable mixture of public and authenticated access and has been designed from the beginning with first-class RESTful web interfaces to support robust API access. Items on ScienceBase can have hierarchical relationships, facilitating the organization of complex or related data. It also supports a seamless transition from project development to data publication, focusing on searchable, accessible, well described datasets for public use and citation.

To expand the usefulness of the ScienceBase platform directly to R workflows, we have designed and implemented an R interface to ScienceBase called **sbtools**. This interface provides scripted R access to ScienceBase to manage metadata and data files, to search the catalog of datasets, and to view and modify data in formats familiar to R users. Here we describe several features of ScienceBase, and how we have implemented the R interface to make them accessible and useful from R.

USGS ScienceBase

To facilitate and encourage data sharing and dissemination, the U.S. Geological Survey has created ScienceBase, an online collaborative scientific data platform (Figure 1). ScienceBase is targeted for use by USGS researchers, their collaborators, and the end users of reviewed and released USGS data products. ScienceBase has four key elements to support collaborative data workflows: 1) Data and metadata cataloging and hosting with options for private, controlled access and fully public sharing

of data and metadata. 2) Central search and data discovery for both data hosted on ScienceBase and externally hosted data. 3) Full web-service support for all core functionality, including standards-based access to specific data types (e.g. geospatial datasets). 4) Research community catalogs to enable the organization of data along collaborative group and organizational boundaries.

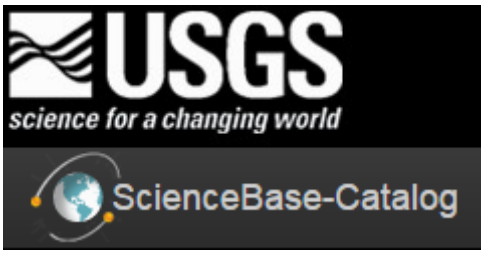


Figure 1: The ScienceBase platform logo

ScienceBase users store data in 'items', where each item is a flexible representation of a dataset and its metadata. The dataset component of an item can be one or more data files in any format or simply an item with descriptive metadata linking a dataset hosted on an external repository. This supports indexing of legacy datasets that are hosted in well known locations (for example, the USGS National Hydrography Dataset). Items are organized into a tree hierarchy (much like the structure of files and folders on a hard drive). This allows data to be intuitively organized by the institution, collaborative group, and/or individual to whom the data belong. At the same time, items can also be assigned identifying tags for rapid search and data discovery across the full ScienceBase catalog.

Search and management of ScienceBase data items can be accomplished through both graphical and scripted interfaces. Manual search, data upload/download, and metadata editing are possible through the ScienceBase website. Automated access to all of these functions is supported by robust RESTful web services and a documented API. The ScienceBase code and infrastructure setup is available upon request from the ScienceBase team <sciencebase@usgs.gov>. Further information and details can be found in the online ScienceBase Documentation.

The sbtools package

In creating **sbtools**, our goal was to allow complete access to the ScienceBase web service API in a flexible, lightweight R package. The package imports a minimal number of external packages to support core functionality. More advanced data access (e.g. geospatial web services) is supported through suggested packages to keep the basic installation requirements minimal for all platforms. Within R, **sbtools** is designed to keep end-user interactions simple despite the underlying complexity of many of the web service calls (e.g. authentication).

Below, we describe briefly the core functions available in **sbtools** and discuss the unique features of ScienceBase that **sbtools** provides R users.

Data access API

The data access functionality of **sbtools** makes it easy to access any public item. Every item in ScienceBase has a unique identifier that can be used for direct access to the item and its associated data and metadata. A lightweight representation of this information is created in R with the `item_get` function.

```
> test_item = item_get("4f4e4b24e4b07f02db6aea14")
> test_item
<ScienceBase Item>
  Title: Coastal-change and glaciological maps of Antarctica
  Creator/LastUpdatedBy:      /
  Provenance (Created / Updated):  2010-10-06T04:25:43Z / 2014-07-21T17:45:42Z
  Children: FALSE
  Item ID: 4f4e4b24e4b07f02db6aea14
  Parent ID: 4f4e4771e4b07f02db47e1e4
```

This representation is defined by **sbtools** as an "sbitem" object, which contains many fields and can be further inspected in the same way as a named list.

```
> names(test_item)
 [1] "link"              "relatedItems"      "id"
 [4] "identifiers"       "title"             "citation"
 [7] "provenance"        "hasChildren"       "parentId"
[10] "contacts"          "webLinks"          "browseCategories"
[13] "browseTypes"       "tags"              "dates"
[16] "facets"            "files"             "distributionLinks"
[19] "previewImage"

> test_item$citation
[1] "Geological Survey (U.S.), 1999-08-05, Coastal-change
and glaciological maps of Antarctica:  Fact SheetCoastal-change and
glaciological maps of Antarctica."
```

On ScienceBase, the hierarchical item tree dictates relationships between items; each item has one parent and potentially many children. **sbtools** allows the user to easily traverse this tree structure. Because ScienceBase allows users to define their internal organization, this heirarchy can take on different meanings for different projects, including conveying data provenance or spatial grouping.

```
#parent ID always available as item attribute
> parent = item_get(test_item$parentId)
> parent
  Title: USGS Publications Warehouse
  Creator/LastUpdatedBy:      /
  Provenance (Created / Updated):  2012-02-29T15:42:41Z / 2014-07-08T21:42:20Z
  Children: TRUE
  Item ID: 4f4e4771e4b07f02db47e1e4
  Parent ID: 4f4e4771e4b07f02db47e1da

#getting sibling items
> item_list_children(parent)
[1] "55b98fbee4b08f6647be5179" "541d45a4e4b0f68901ec30ef"
[3] "55b361b3e4b09a3b01b5daad" "53516ef9e4b05569d8059f34"
[5] "4f4e4ab2e4b07f02db66f5e3" "5351704ee4b05569d805a2e4"
```

ScienceBase items may have data or metadata files attached to them. You can list and download attached files directly using **sbtools**.

```
#returns names of files attached to item
> item_list_files(test_item)
                             fname size  url
  1 metadata6644450227216673613.xml 1742  https://www.sciencebase.gov/[truncated]

#returns local path to downloaded files
> item_file_download(test_item, dest_dir = tempdir())
[1] "\\path\\to\\file\\RtmpgBV2fn/metadata6644450227216673613.xml"
```

ScienceBase has special functionality for certain data types. One example is spatial data. When spatial data are uploaded to ScienceBase and appropriate metadata is included, they can be accessed using Open Geospatial Consortium (OGC) web services. **sbtools** includes functionality to access Web Feature Service (WFS) when available. Once retrieved, spatial data are stored as **sp** spatial objects, which are easily manipulated and visualized (Figure 2). Note: Some figure formatting details omitted from code below for simplicity. See `demo('figure_map_code',package='sbtools')` for the full example.

```
#Load non-sbtools-required but useful mapping packages
library(maps)
library(sp)
#an item with an included OGC WFS service
layer = item_get_wfs('55e372b9e4b05561fa208212')
map('state', regions = 'iowa')
plot(layer, add = TRUE)
```

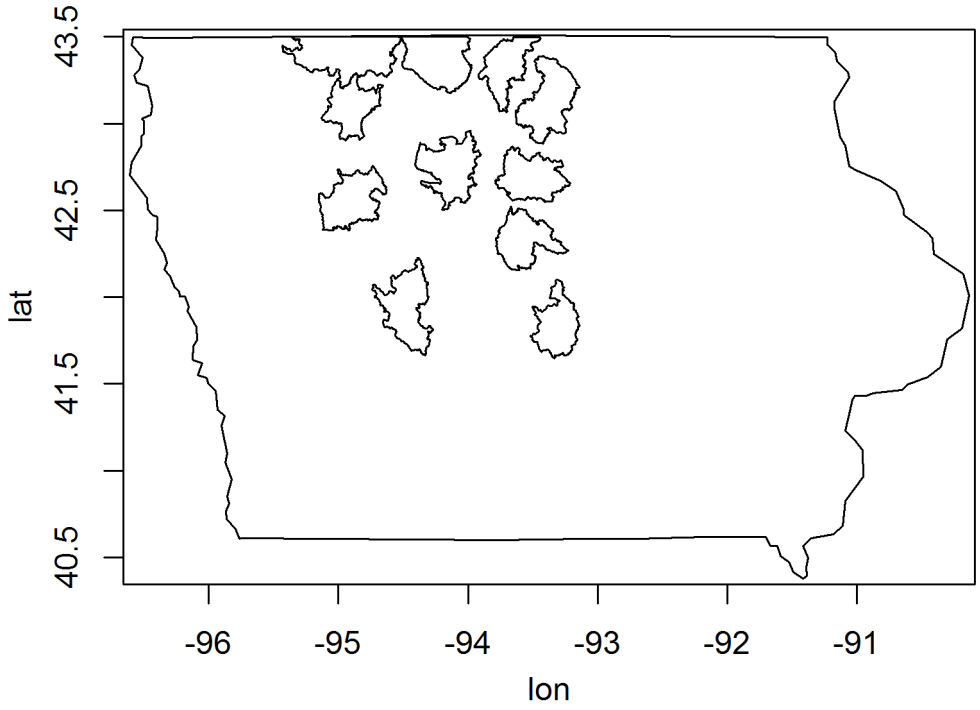


Figure 2: An example spatial dataset from ScienceBase. Shows high priority regions in Iowa containing shallow lakes and marshes to create management emphasis areas for migratory water fowl habitat.

Search API

To support advanced and powerful data discovery, all datasets in ScienceBase are indexed and made available through a flexible search interface. **sbtools** offers several query functions with different levels of search specificity, from cross-metadata simple text search to low-level, metadata specific search functionality.

In **sbtools**, query functions are included to support the most common query types. Included functions are available to search based on free-text, project folder, date-time range, geospatial bounding box, Digital Object Identifier (DOI), and data type. To save space in the examples below, we use the `limit` parameter to limit all queries to the first one or two results. Further details for each query type are available in the package documentation.

The ScienceBase free-text search is simple to use and generic as it searches across all of an item's text-based metadata fields. Free-text search can find specific text strings in almost all metadata fields, including but not limited to filenames, summary fields, citation, contacts, and authors. Spatial reference metadata is an example of a field not searched through the free-text search. Using **sbtools**, a free-text search can be run with the `query_sb_text` function.

```
> query_sb_text('Antarctica', limit=1)
[[1]]
<ScienceBase Item>
  Title: Antarctica. Erratic near camp 13. December 12, 1977.
  Creator/LastUpdatedBy:      /
  Provenance (Created / Updated):  2013-07-09T15:38:45Z / 2016-03-02T23:55:43Z
  Children: FALSE
  Item ID: 51dc2e85e4b0f81004b79cf0
  Parent ID: 519ba0a3e4b0e4e151ef5dd9
```

Items can also be queried according to their position in the hierarchical item tree. For example, projects funded by USGS Climate Science Centers each have sections on ScienceBase where project information is stored. Using the web interface (and some user knowledge), we first found the ScienceBase ID of the Northeast Climate Science Center community folder. Then, **sbtools** was used to look for specific items in that commuinty. Below is an example of looking for items containing any reference to "Lake Superior" under the Northeast Climate Science Center community.

```
#Look for items referencing "Lake Superior" under the NE Climate Science Center projects
> query_item_in_folder("Lake Superior", folder="4f8c648de4b0546c0c397b43", limit=2)
[[1]]
<ScienceBase Item>
  Title: The Role of Lake-Dotted Landscapes in Regional Climate Change [...]
  Creator/LastUpdatedBy:      /
  Provenance (Created / Updated):  2013-12-31T17:46:16Z / 2013-12-31T17:46:16Z
  Children: FALSE
  Item ID: 52c302e8e4b040b25da9d35a
  Parent ID: 51db0ebce4b010c7f6a814bf

[[2]]
<ScienceBase Item>
  Title: The influence of land use and [...] in Lake Superior tributaries
  Creator/LastUpdatedBy:      /
  Provenance (Created / Updated):  2013-12-31T17:45:51Z / 2013-12-31T17:45:51Z
  Children: FALSE
  Item ID: 52c302cfe4b040b25da9ce3c
  Parent ID: 51db0ebce4b010c7f6a814bf
```

There are a several more useful query types that can be performed simply with **sbtools**. A few examples include date-time range, (`query_sb_date()`), data type (`sb_datatypes()` and `query_sb_datatype`), and Digial Object Identifier (DOI; `query_sb_doi()`).

```
#Query recently updated items
> query_sb_date(Sys.time()-as.difftime(7, units="days"), Sys.time(),
+    date_type='lastUpdated', limit=1)
[[1]]
<ScienceBase Item>
  Title: US Topo
  Creator/LastUpdatedBy:      /
  Provenance (Created / Updated):  2012-03-05T22:46:14Z / 2016-02-01T12:11:58Z
  Children: TRUE
  Item ID: 4f554236e4b018de15819c85
  Parent ID: 4f552e93e4b018de15819c51


#Query for USGS Reports
> query_sb_datatype('Report', limit = 1)
[[1]]
<ScienceBase Item>
  Title: Final Memo for Structured decision-making to facilitate multi-stakeholder
  coastal conservation and restoration under climate change uncertainties: case study
  on barrier islands of the northern Gulf of Mexico
```

```
  Creator/LastUpdatedBy:      /
  Provenance (Created / Updated):   /
  Children:
  Item ID: 565e07b3e4b071e7ea5435d0
  Parent ID: 5224e64fe4b0e4746d62af85

#Query for DOI
> query_sb_doi('10.5066/F7M043G7')
[[1]]
<ScienceBase Item>
  Title: 2013 Raw Ground Penetrating Radar Data on Alaska's Glaciers
  Creator/LastUpdatedBy:      /
  Provenance (Created / Updated):  2015-06-15T16:55:03Z / 2015-12-15T20:39:06Z
  Children: TRUE
  Item ID: 557f0367e4b023124e8ef621
  Parent ID: 5474ec49e4b04d7459a7eab2
```

Items with a geospatial component to their data or metadata can be queried using a Lat/Lon bounding box with the function `query_sb_spatial()`. The bounding box may be directly specified with coordinates or indirectly specified by supplying another spatial object whose bounding box should be used. Because spatial functionality requires packages beyond those imported by **sbtools** by default, the **xml2**, **sp** and **rgdal** packages must be installed to use these functions.

```
#specify the latitude and longitude points to define the bounding box range.
# This is simply bottom left and top right points
> query_sb_spatial(long=c(-104.4, -95.1), lat=c(37.5, 41.0), limit=1)
[[1]]
<ScienceBase Item>
  Title: National Fish Habitat Partnership Data System
  Creator/LastUpdatedBy:      /
  Provenance (Created / Updated):  2012-02-29T15:42:43Z / 2015-11-12T20:19:30Z
  Children: TRUE
  Item ID: 4f4e4773e4b07f02db47e241
  Parent ID: 4f4e4760e4b07f02db47df9c

##You can also use the bounding box of an sp spatial data object
#grab an sp object from a pre-determined ScienceBase Item
> layer = item_get_wfs('55e372b9e4b05561fa208212')

#get items in that bounding box
> query_sb_spatial(layer, limit = 1)

[[1]]
<ScienceBase Item>
  Title: USGS Denver Library Photographic Collection
  Creator/LastUpdatedBy:      /
  Provenance (Created / Updated):  2013-05-21T16:28:19Z / 2016-01-04T15:44:20Z
  Children: TRUE
  Item ID: 519ba0a3e4b0e4e151ef5dd9
  Parent ID: 4f4e4771e4b07f02db47e1da
```

Lastly, **sbtools** offers advanced access to all query capabilities through the `query_sb()` function. `query_sb()` provides a convenient wrapper that allows the user to supply a list of query parameters (options described in the documentation), submits that query to ScienceBase and parses the output into a list of "`sbitem`" objects. When necessary, `query_sb()` also does proper result paging when the requested return length (limit) is over 1000 items. All advanced search options can be experimented with via the online advanced search interface.

```
# query_sb can be used for combined search criteria
> query_sb(list(q = "water", folderId = '504216b9e4b04b508bfd337d',
```

```
                browserCategory = 'Image'), limit=1)
[[1]]
<ScienceBase Item>
  Title: Encyclopedia of Water Science
  Creator/LastUpdatedBy:      /
  Provenance (Created / Updated):  2013-04-18T15:06:38Z / 2013-04-18T15:06:38Z
  Children: FALSE
  Item ID: 51700bfee4b05024ef3cd4ef
  Parent ID: 504216b9e4b04b508bfd337d
```

Combining both the query functionality and direct spatial data access through WFS web services, we can quickly discover and explore available datasets on ScienceBase. For example, if we wanted to discover what faultline spatial data are available on ScienceBase, we can combine **sbtools** query and data access functionality. In the below example, we query for items with the word "faults" in their description which also expose data through an OGC WFS web service. We plot all datasets together to general geographic locations and data type/completeness (Figure 3). We see datasets covering much of the western United States and off the west coast. Small datasets are scattered around the gulf coast and other areas of the U.S., but none expose the raw faultline data so more discovery may be required.

Note: Some formatting details of code omitted from code below for simplicity. See `demo('figure_fault_code',package='sbtools')` for full example.

```
#Source non-sbtools-required but useful mapping packages
library(sp)
library(maps)

faults = query_sb(list(q = "faults", browseType = "OGC WFS Layer"), limit = 20)

map('usa')
for(i in 1:length(faults)){
    layer = item_get_wfs(faults[[i]]$id)
    layer = spTransform(layer, CRS('+proj=longlat +datum=WGS84'))
    plot(layer, add=TRUE, col='red')
}
map.axes()
```

ScienceBase authentication

In addition to the large collection of open, reusable datasets and useful metadata ScienceBase has to offer, it is also a platform for sharing and collaboration on private, in-progress data available only through authenticated access. To enable private data contribution and access, **sbtools** has built-in support for persistent authentication of R sessions.

In **sbtools**, users can log into ScienceBase with their ScienceBase username and password using the function `authenticate_sb()`. To prevent plain-text passwords from being saved in the R command history, when using RStudio, the password can be typed into a pop-up window. The core R environment displays a terminal input interface for the password. **sbtools** only stores the authenticated session, thereby maintaining the confidentiality of the user's credentials. ScienceBase sessions remain active for roughly one hour and are renewed each time a request is made. **sbtools** makes a best effort to supply the user with meaningful error messages when a session may have expired.

```
#to start an authenticated session
> authenticate_sb('username@usgs.gov') #password entered into pop-up window
> is_logged_in()
[1] TRUE
> session_logout()
> is_logged_in()
[1] FALSE
```

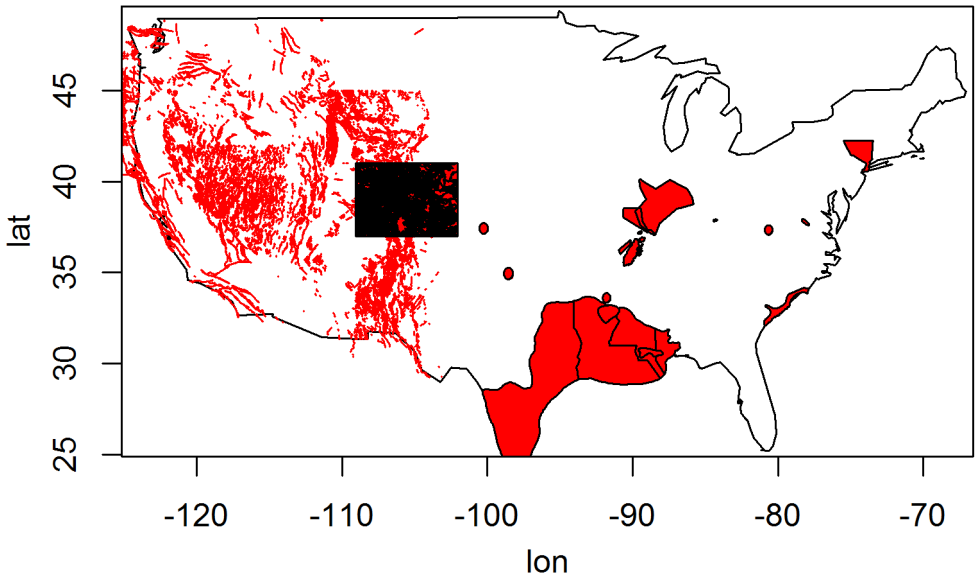


Figure 3: An example of querying and discovering multiple spatial datasets on ScienceBase. In this case, faultlines across the U.S. The variability in data formats (some lines, some polygons) is intentionally included to show both data variability, and quick discovery of different data types across various regions.

Most functions in **sbtools** can be used both anonymously and when authenticated. This includes all data retrieval and query functions. The behavior of these functions depends on the user's authentication status and access permissions. For example, when trying to access a private item using `item_get()`, you must be authenticated or you will receive an error that the item is missing. To maintain privacy, ScienceBase does not differentiate between a missing item and an item you lack permission to access. Search is also dependent on authentication status. When querying items with `query_sb()`, public items are always visible while private items are invisible unless authenticated.

```
#Attempt to get a private item without authentication
> item_get('55de0027e4b0518e354dfcf0')
 Error: Item not found for ID=55de0027e4b0518e354dfcf0. Either the
 item does not exist or the item is secured and requires authentication to access.

#Get private item while authenticated
> authenticate_sb('username@usgs.gov')
> item_get('55de0027e4b0518e354dfcf0')
 <ScienceBase Item>
  Title: Example Private Item
  Creator/LastUpdatedBy:     username@usgs.gov / username@usgs.gov
  Provenance (Created / Updated):  2015-08-26T18:06:31Z / 2015-12-30T14:59:27Z
  Children: FALSE
  Item ID: 55de0027e4b0518e354dfcf0
  Parent ID: 54257d8fe4b0e641df8b50af

#Search results include user's private items when authenticated
> query_sb(list(q = 'username@usgs.gov'))
 <ScienceBase Item>
  Title: Example Private Item
  Creator/LastUpdatedBy:     username@usgs.gov / username@usgs.gov
  Provenance (Created / Updated):  2015-08-26T18:06:31Z / 2015-12-30T14:59:27Z
  Children: FALSE
  Item ID: 55de0027e4b0518e354dfcf0
  Parent ID: 54257d8fe4b0e641df8b50af

#Search results hide private items when not authenticated
> session_logout()
```

```
> query_sb(list(q = 'username@usgs.gov'))
list()
```

The authentication status can be quickly checked and updated with a few helper functions.

```
#See if user is authenticated to SB
> is_logged_in()
[1] TRUE

#Get details of authenticated session
> session_details()
$fullDisplayName
[1] "User McUseface [username@usgs.gov]"
$isLoggedIn
[1] TRUE
$displayName
[1] "User McUseface"
$email
[1] "username@usgs.gov"
$username
[1] "username@usgs.gov"

#Renew a session to prevent expiration after 1 hour
> session_renew()
```

Data editing and upload API

For authenticated users, **sbtools** can support the full data lifecycle. This includes the creation, editing and removal of items. Because item editing and creation cannot be done anonymously, this functionality only works while authenticated.

To create new items, the only required input to `item_create` is "title". The "Parent Item" may also be specified. New items, by default, inherit the privacy settings of their parent item. All users have a personal home folder (called "My Items" on the ScienceBase website) that serves as the default parent item for new items. The unique identifier of a user's home folder can be retrieved with the function `user_id()`.

```
#create new item, by default under "My Items" parent
> new_item = item_create(title = 'new test item')
> new_item
<ScienceBase Item>
  Title: new test item
  Creator/LastUpdatedBy:     username@usgs.gov / username@usgs.gov
  Provenance (Created / Updated):  2016-01-27T19:48:28Z / 2016-01-27T19:48:28Z
  Children: FALSE
  Item ID: 56a91f0ce4b0b28f1184dda8
  Parent ID: 54257d8fe4b0e641df8b50af
```

Once an item is created, an authenticated user can edit the metadata or attach data files to that item.

```
#give the item a new title
> edited_item = item_update(new_item, list(title = 'new updated item'))
> edited_item
<ScienceBase Item>
  Title: new updated item
  Creator/LastUpdatedBy:     lwinslow@usgs.gov / lwinslow@usgs.gov
  Provenance (Created / Updated):  2016-01-27T19:48:28Z / 2016-01-27T19:50:21Z
```

```
  Children: FALSE
  Item ID: 56a91f0ce4b0b28f1184dda8
  Parent ID: 54257d8fe4b0e641df8b50af

#append files to the item
> item_list_files(new_item)
data frame with 0 columns and 0 rows

> item_append_files(edited_item, 'test.dat')
> item_list_files(edited_item)
      fname size     url
1 README.md 1282     https://www.sciencebase.gov/[long URL truncated]
```

Functions are also provided to modify and delete attached files and to delete entire items.

```
#list currently attached files
> item_list_files(edited_item)
      fname size     url
1 README.md 1282     https://www.sciencebase.gov/[long URL truncated]

#selectively replace files. all = FALSE replaces files one by one;
#otherwise all files are removed before uploading the new files
> item_replace_files(new_item, 'README.md', all = FALSE)
> item_list_files(edited_item)
      fname size     url
1 README.md 608     https://www.sciencebase.gov/[long URL truncated]

#delete item. use with caution; there is no confirmation check
> item_rm(edited_item)
> item_get(edited_item)
Error: Item not found for ID=56a91f0ce4b0b28f1184dda8. Either the item does
not exist or the item is secured and requires authentication to access.
```

SB item identifiers

One advanced feature of ScienceBase is the ability to assign any number of custom item identifiers to items. The custom identifiers are made up of three parts: Scheme, Type and Key. Combined, these create a unique identifier. There are some standard Schemes used in ScienceBase. For example, DOIs (Digital Object Identifiers) are stored as item identifiers with the Scheme "https://www.sciencebase.gov/vocab/category/item/identifier" and Type "DOI".

sbtools can edit and query custom item identifiers using `item_update_identifier` and `query_item_identifier`, respectively.

```
#create two items and assign custom identifiers
> ident_item = item_create(title = 'test data')
> item_update_identifier(ident_item, scheme = 'proj2', type = 'data', key = 'dataset1')
> ident_item = item_create(title = 'test publication')
> item_update_identifier(ident_item, scheme = 'proj2', type = 'publication', key = 'pdf1')

#query for created item
> query_item_identifier(scheme = 'proj2', type = 'publication', key = 'pdf1')
             title                        id
1 test publication 56a9371ee4b012c193aa3d65
```

The three-part identifier can be especially useful as a way to organize and access project data. For example, all items within the same project could be created with the same scheme and differing types or keys, allowing users to query items within the project using custom tags that are meaningful to that project.

```
#query all items in 'proj2'
> query_item_identifier(scheme = 'proj2')
             title                       id
1 test publication 56a9371ee4b012c193aa3d65
2        test data 56a93777e4b012c193aa3d68

#query just data items for a project
> query_item_identifier(scheme = 'proj2', type = 'data')
      title                      id
1 test data 56a93777e4b012c193aa3d68
```

Summary

As many real-world projects have demonstrated, R is an excellent tool for collaborative and reproducible research projects (Gandrud, 2013a). New packages are frequently opening up access to large, open datasets (Bowman and Lees, 2015; Bowman, 2014). Flexible and powerful ways to share and collectively work with common datasets could enable new modes of collaboration and diverse data discovery. To help fill this gap, we have created **sbtools**. The **sbtools** package gives the user programmatic access to the cloud-based data and metadata storage of USGS ScienceBase and allows all researchers direct query and download capabilities to free, public data on ScienceBase. **sbtools** enables rapid and reproducible access to one of the single largest repositories of Earth-science data and an advanced cloud-based data collaboration platform.

Acknowledgments

This work was supported by the U.S. Geological Survey (USGS), Office of Water Information and by funding from the USGS Community for Data Integration (funding title: sbtools: An R package for ScienceBase). We would like to thank Drew Ignizio, Marian Talbert and the two journal reviewers for their careful reviews and feedback on the manuscript and package. Any use of trade, firm, or product names is for descriptive purposes only and does not imply endorsement by the U.S. Government. (

Bibliography

C. Boettiger, S. Chamberlain, K. Ram, and E. Hart. *rfigshare: An R Interface to 'figshare'*. URL https://github.com/ropensci/rfigshare. R package version 0.3.7.99. [p387]

D. C. Bowman. *rFDSN: Get Seismic Data from the International Federation of Digital Seismograph Networks*, 2014. URL https://CRAN.R-project.org/package=rFDSN. R package version 0.0.0. [p397]

D. C. Bowman and J. M. Lees. Near real time weather and ocean model data access with rNOMADS. *Computers & Geosciences*, 78:88–95, 2015. doi: 10.1016/j.cageo.2015.02.013. [p397]

S. Chamberlain. *ckanr: Client for the Comprehensive Knowledge Archive Network ('CKAN') 'API'*. URL https://github.com/ropensci/ckanr. R package version 0.1.0. [p387]

A. Delcambre. Github - New File Size Limits, 2013. URL https://github.com/blog/1533-new-file-size-limits. [p387]

C. Gandrud. *Reproducible Research with R and RStudio*. Chapman and Hall/CRC, Boca Raton, FL, USA, 2013a. ISBN 9781466572843. [p397]

C. Gandrud. GitHub: A tool for social data set development and verification in the cloud. *SSRN Electronic Journal*, 20:1–10, 2013b. ISSN 1556-5068. doi: 10.2139/ssrn.2199367. URL http://www.ssrn.com/abstract=2199367$\delimiter"026E30F$nhttp://papers.ssrn.com/abstract=2199367. [p387]

M. Jones, R. Nahf, C. Jones, C. Boettiger, L. Walker, S. Chamberlain, E. Hart, J. Read, and P. Slaughter. *dataone: R interface to the DataONE REST API*, 2013. URL https://CRAN.R-project.org/package=dataone. R package version 1.1.0.9008. [p387]

G. King. An Introduction to the Dataverse Network as an Infrastructure for Data Sharing. *Sociological Methods & Research*, 36(2):173–199, 2007. ISSN 0049-1241. doi: 10.1177/0049124107306660. [p387]

T. J. Leeper. *dvn: Access to The Dataverse Network APIs*, 2013. R package version 0.3.3. [p387]

W. K. Michener, S. Allard, A. Budden, R. B. Cook, K. Douglass, M. Frame, S. Kelling, R. Koskela, C. Tenopir, and D. A. Vieglais. Participatory design of DataONE-Enabling cyberinfrastructure for the biological and environmental sciences. *Ecological Informatics*, 11:5–15, 2012. ISSN 15749541. doi: 10.1016/j.ecoinf.2011.08.007. URL http://dx.doi.org/10.1016/j.ecoinf.2011.08.007. [p387]

J. Singh. FigShare. *Journal of Pharmacology and Pharmacotherapeutics*, 2(2):138, 2011. ISSN 0976-500X. doi: 10.4103/0976-500X.81919. URL http://www.jpharmacol.com/text.asp?2011/2/2/138/81919. [p387]

J. Winn. Open data and the academy: An evaluation of CKAN for research data management. In *IASSIST 2013, Cologne*, 2013. URL http://eprints.lincoln.ac.uk/9778/. [p387]

Luke A Winslow
U.S. Geological Survey, Office of Water Information
Middleton, Wisconsin
USA
lwinslow@usgs.gov

Scott Chamberlain
rOpenSci / University of California, Berkeley
Berkeley, California
USA
scott@ropensci.org

Alison P Appling
U.S. Geological Survey, Office of Water Information
Middleton, Wisconsin
USA
aappling@usgs.gov

Jordan S Read
U.S. Geological Survey, Office of Water Information
Middleton, Wisconsin
USA
jread@usgs.gov

Conference Report: useR! 2016

by Joseph Rickert

Overview

The 12th international R user conference, useR! 2016, took place at Stanford University, Stanford CA from June 27 through June 30th. Hosted by the Stanford University Department of Statistics and the Stanford Libraries, the conference took place at the Frances Arrillaga Alumni Center, on the surrounding lawns and in several adjacent buildings. The floor to ceiling windows of the larger conference rooms, the garden locations for coffee and meals and the beautiful weather contributed to making the event a classic California experience.

Originally planned for 750 people, a slight increase over the 660 attendee total for last year's conference in Denmark, useR! 2016 sold out completely during the first few weeks. Although the final attendee list eventually topped out at just over 900 people, it nevertheless excluded many from both academia and industry who were seeking tickets. To mitigate the disappointment, the conference organizers arranged to "live stream" the keynote sessions over the internet and to record many of the contributed talks. Many thanks to Microsoft Corporation which provided the expertise and financing for the video recording, and to many other corporate sponsors who made possible student scholarships, daily free lunches, a continuous flow of coffee and fruit juices, and a social program that included a cocktail reception and a Hornblower Yacht cruise on the San Francisco Bay.

The Gordon and Betty Moore foundation helped fund 17 Diversity Scholarship awards overseen by a committee consisting of Scott Chamberlain, Amy Lee, Gabriela de Queiroz and Karthik Ram (chair). In addition, the American Statistical Association provided funds to award $2,500 each to two outstanding young useRs, Helen Ogden (University of Warwick) and Tong He (Simon Fraser University) chosen by the Program Committee.

The program consisted of 18 pre-conference tutorials, 6 invited talks, 146 oral presentations, 45 lightning talks and 60 poster sessions.

Pre-conference Tutorials

The pre-conference tutorials were free and open to all attendees.

- Regression Modeling Strategies and the **rms** Package - Frank Harrell
- Using Git and GitHub with R, RStudio, and R Markdown - Jennifer Bryan
- Effective Shiny Programming - Joe Cheng
- Missing Value Imputation with R - Julie Josse
- Extracting data from the web APIs and beyond - Scott Chamberlain, Garrett Grolemund and Karthik Ram
- Ninja Moves with data.table - Learn by Doing in a Cookbook Style Workshop - Matt Dowle and Arun Srinivasan
- Never Tell Me the Odds! Machine Learning with Class Imbalances - Max Kuhn
- MoRe than woRds, Text and Context: Language Analytics in Finance with R - Sanjiv Das and Karthik Mokashi
- Handling and Analyzing Spatial, Spatiotemporal and Movement Data - Edzer Pebesma
- Machine Learning Algorithmic Deep Dive- Erin LeDell
- Introduction to SparkR- Hossein Falaki and Shivaram Venkataraman

- Using R with Jupyter Notebooks for Reproducible Research - Andrie de Vries and Micheleen Harris
- Understanding and Creating Interactive Graphics - Claus Thorn Ekstrøm and Toby Dylan Hocking
- Genome-Wide Association Analysis and Post-Analytic Interrogation with R - Andrea S. Foulkes
- An Introduction to Bayesian Inference using R Interfaces to Stan - Ben Goodrich
- Small Area Estimation with R - Virgilio Gómez Rubio
- Dynamic Documents with R Markdown- Yihui Xie

Invited talks

The invited, plenary talks began with a retrospective look at the development of the S and R languages, discussed topics concerned with good programming practice and touched on topics essential to the developing field of Data Science.

- Forty years of S - Richard Becker
- Literate Programming - Donald Knuth
- Towards a grammar of interactive graphics - Hadley Wickham
- Flexible and Interpretable Regression Using Convex Penalties - Daniela Witten
- Statistical Thinking in a Data Science Course - Deborah Nolan
- RCloud - Collaborative Environment for Visualization and Big Data Analytics - Simon Urbanek

Contributed Sessions

The contributed talks were organized into 5 parallel tracks with sessions devoted to: Bayesian Statistics, Bioinformatics, Case Studies, Databases, Generalized Mixed Models, Graphics, Packages and Development, Performance, R in Business, R and Other Languages, Regression, Reproducible Research, Spatial Statistics, Statistical Methods, Statistics and Big Data, Teaching and sessions devoted to our sponsors, miscellaneous talks organized under Kaleidoscope sessions and lightning talks.

Conference Organizers

The strong, diverse technical program was the work of program committee members Jenny Bryan, Dianne Cook, Peter Dalgaard, Dirk Eddelbuettel, Susan Holmes, Torsten Hothorn, Julie Josse (Chair), Patrick Mair, Jeroen Ooms, Hilary Parker, Hana Ševčíková, Torben Tvedebrink and Heather Turner.

The conference would not have been possible without the tireless work of Balasubramanian Narasimhan who led the organizing committee: John Chambers, Sandrine Dudoit, Trevor Hastie, Susan Holmes, Simon Jackman, Olivia Lau, Nicholas Lewin-Koh, Norman Matloff, Jacqueline Meulman, Balasubramanian Narasimhan, Karthik Ram, Joseph Rickert and Duncan Temple Lang. The cheerful presence and help provided by student volunteers chosen from the R community helped make the conference a pleasant experience for all attendees.

Additional Information

useR!2106 website http://user2016.org/

Video recordings https://channel9.msdn.com/Events/useR-international-R-User-conference/useR2016

Corporate sponsors http://user2016.org/#sponsors

Tutorial perspective http://blog.revolutionanalytics.com/2016/06/the-user-2016-tutorials.html

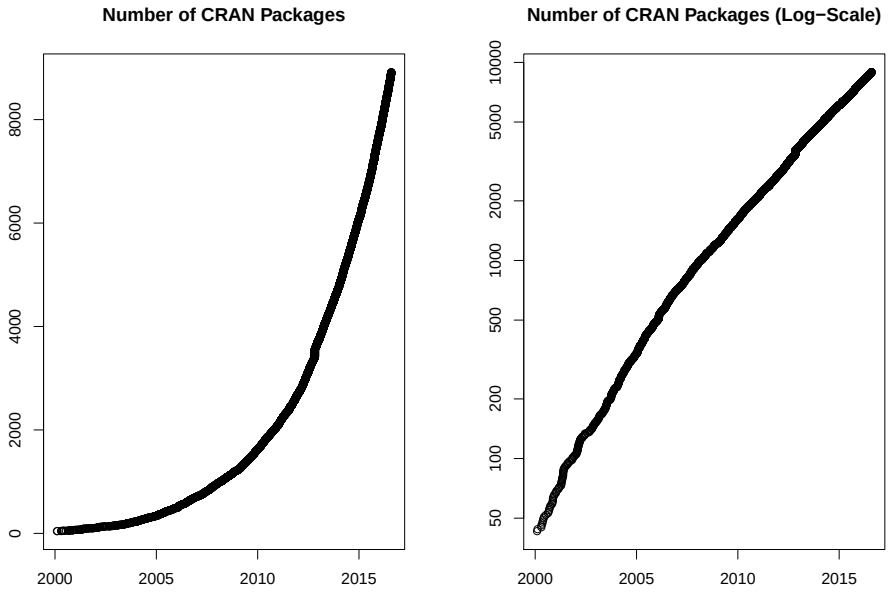
Package perspective http://blog.revolutionanalytics.com/2016/06/the-r-packages-of-user-2016.html

Changes on CRAN

2015-12-01 to 2016-07-31

by Kurt Hornik and Achim Zeileis

In the past 8 months, 1322 new packages were added to the CRAN package repository. 43 packages were unarchived, 48 archived, 1 package had to be removed. The following shows the growth of the number of active packages in the CRAN package repository:



At the R Foundation's General Assembly after UseR! 2016 in Stanford, the CRAN team asked for help, in particular for processing package submissions. Dirk Eddelbüttel, Duncan Murdoch, Deepayan Sarkar, and Duncan Temple Lang volunteered. Duncan Murdoch is already actively processing incoming CRAN submissions; expect other changes over the coming months.

New packages in CRAN task views

ChemPhys **EEM**, **titrationCurves**, **webchem**.

Cluster **evclust**, **genie**, **treeClust**.

Distributions **EnvStats**, **KScorrect**, **bridgedist**, **extraDistr**, **extremefit**, **marg**, **mclust**.

Econometrics **rUnemploymentData**, **wbstats**.

Finance **FRAPO**, **XBRL**, **bootTimeInference**, **derivmkts**, **finreportr**, **obAnalytics**.

HighPerformanceComputing **LaF**, **RcppParallel**, **doFuture**, **future.BatchJobs**, **gpuR**, **h2o**, **randomForestSRC**, **sprint**.

MachineLearning **OneR**, **SIS**, **SuperLearner**, **evclass**, **h2o**, **hdm**.

MetaAnalysis **altmeta**, **bayesmeta**, **bmeta**, **gmeta**, **hetmeta**, **metansue**, **weightr**.

NumericalMathematics **conicfit**, **madness**, **matrixcalc**, **permutations**.

OfficialStatistics **convey**, **icarus**.

Optimization **cmaesr**, **nlmrt**, **parma**, **psoptim**, **rCMA**, **rLindo**, **scs**, **smoof**.

Psychometrics **ShinyItemAnalysis**, **blavaan***, **bpca**, **difNLR**, **dualScale**, **metaSEM**, **quickpsy**, **wCorr**.

ReproducibleResearch **DT**, **HTMLUtils**, **Kmisc**, **RefManageR**, **ReporteRs**, **SortableHTMLTables**, **apaStyle**, **archivist**, **checkpoint**, **compareGroups**, **connect3**, **formatR**, **formattable**, **highlight**, **highr**, **htmlTable**, **htmltools**, **humanFormat**, **kfigr**, **knitLatex**, **knitcitations**, **latex2exp**, **lazyWeave**, **lubridate**, **miniCRAN**, **packrat**, **prettyunits**, **rbundler**, **resumer**, **rmarkdown**, **rprintf**, **tufterhandout**, **ztable**.

Robust **roahd**.

Spatial **HSAR**, **ProbitSpatial**, **RNetCDF**, **S2sls**, **SpatialPosition**, **Watersheds**, **cartography**, **cleangeo**, **diseasemapping**, **gdalUtils**, **geoaxe**, **geostatsp**, **igraph**, **ipdw**, **lawn**, **lctools**, **magclass**, **mapmisc**, **mapview**, **ncdf4**, **quickmapr**, **recmap**, **shp2graph**, **spanel**, **statebins**, **stplanr**.

SpatioTemporal **VTrack**, **trackeR**.

TimeSeries **ForecastCombinations**, **M4comp**, **VARsignR**, **ZRA**, **carx**, **sleekts**, **stlplus**, **tsPI**.

WebTechnologies **ApacheLogProcessor**, **AzureML**, **FastRWeb**, **GAR**, **RAdwords**, **RGoogleFit**, **ROpenWeatherMap**, **RSclient**, **RYandexTranslate**, **RZabbix**, **Rblpapi**, **Rexperigen**, **Rmonkey**, **Rserve**, **V8**, **WikiSocio**, **WikidataR**, **WufooR**, **abbyyR**, **aws.signature**, **backblazer**, **bigrquery**, **boxr**, **captr**, **clarifai**, **curlconverter**, **cymruservices**, **ddeploy**, **discgolf**, **fbRads**, **fitbitScraper**, **fitcoach**, **genderizeR**, **geocodeHERE**, **git2r**, **gitlabr**, **googlesheets**, **graphTweets**, **gsheet**, **httpcache**, **httping**, **instaR**, **jug**, **livechatR**, **longurl**, **lucr**, **mime**, **oai**, **osrm**, **pdftables**, **rLTP**, **randNames**, **rdatacite**, **request**, **restimizeapi**, **rgeolocate**, **rio**, **rorcid**, **rrefine**, **rvest**, **searchConsoleR**, **sendmailR**, **soql**, **telegram**, **threewords**, **tidyjson**, **transcribeR**, **tweet2r**, **urlshorteneR**, **webreadr**, **webshot**, **wikipediatrend**, **xml2**, **yummlyr**.

(* = core package)

Kurt Hornik
WU Wirtschaftsuniversität Wien, Austria
Kurt.Hornik@R-project.org

Achim Zeileis
Universität Innsbruck, Austria
Achim.Zeileis@R-project.org

News from the Bioconductor Project

by Bioconductor Core Team

The Bioconductor project provides tools for the analysis and comprehension of high-throughput genomic data. The 1211 software packages available in Bioconductor can be viewed at `http://bioconductor.org/packages/`. Navigate packages using 'biocViews' terms and title search. Each package has an html page with a description, links to vignettes, reference manuals, and usage statistics. Start using Bioconductor version 3.3 by installing R 3.3.1 and evaluating the commands

```
source("https://bioconductor.org/biocLite.R")
biocLite()
```

Install additional packages and dependencies, e.g., **AnnotationHub**, with

```
source("https://bioconductor.org/biocLite.R")
biocLite("AnnotationHub")
```

Continued availability of Bioconductor Docker and Amazon images provides a very effective on-ramp for power users to rapidly obtain access to standardized and scalable computing environments.

Bioconductor 3.3 Release Highlights

Bioconductor 3.3 was released on 4 April, 2016. It is compatible with R 3.3 and consists of 1211 software packages, 293 experiment data packages, and 916 up-to-date annotation packages. There are 107 new software packages and many updates and improvements to existing packages. The release announcement includes descriptions of new packages and updated NEWS files provided by package maintainers.

Our collection of microarray, transcriptome and organism-specific *annotation packages* use the 'select' interface (keys, columns, keytypes) to access static information on gene annotations (**org.*** packages) and gene models (**TxDb.*** packages); these augment packages for querying web-based resources. The **AnnotationHub** continues to complement our traditional offerings with diverse whole genome annotations from Ensembl, ENCODE, dbSNP, UCSC, and elsewhere; example uses are described in the AnnotationHub How-To vignette.

User support

The Bioconductor project web site helps orient users and developers to the project. It includes essential information for software installation, detailed landing pages for each package (e.g., `https://bioconductor.org/packages/GenomicRanges`) including links to current manuals and vignettes, extensive training material, and links to the current literature. A recent innovation has been the development of the Bioconductor F1000 publishing channel for academic publication of work flows and other extended software use cases.

The project support site is a question-and-answer forum where users can easily search for existing solutions or pose specific questions about use of Bioconductor packages. The support site is quite active, with expert responses often within a matter of hours. It is very helpful, when asking about error messages, to ensure that your Bioconductor installation is correct (using `BiocInstaller::biocValid()`) and current (include the output of `sessionInfo()` in your question), that the question includes code chunks that someone else can evaluate to reproduce the problem (e.g., using code or data from example pages of package manuals), and that the error message and `traceback()` output are included.

Bioconductor holds an annual user conference each summer, this year in conjunction with UseR! 2016. Conference resources (talks and workshops) are available.

Developer support

A very natural progression in the R and Bioconductor community is from user to package developer, transforming your knowledge and domain expertise into software that others can use. The Bioconductor web site includes developer resources to help this transition. The Bioconductor developer mailing list provides a forum dedicated to developer-related questions.

New packages are now submitted to Bioconductor using an open review model. Prospective authors develop their package and, when ready, open an issue on the public Contributions github repository. Packages are then built and checked across Linux, Mac, and Windows platforms for conformance to R (`R CMD check`) and Bioconductor (using the **BiocCheck** package) standards. Once the package is in good shape, a member of the Bioconductor core team performs a preview of the package. The preview identifies technical issues that are not easy to detect automatically.

A key strength of the Bioconductor project is the use of well-defined objects (especially from the **GenomicRanges** infrastructure) to represent data; this encourages software re-use and enables end-user interoperability between packages. For this reason, the technical review often leads to suggestions for data representations and interfaces that use Bioconductor objects rather than general-purpose containers such as a `data.frame`.

Forthcoming activities

Forthcoming Bioconductor events include an Asian workshop workshop and developer meeting (3-4 November, Brisbane, Australia) and European developer conference (6-7 December, Basel, Switzerland) developer conferences, as well as global training opportunities.

The next Bioconductor release will occur in October, 2016.

Bioconductor Core Team
Biostatistics and Bioinformatics
Roswell Park Cancer Institute, Buffalo, NY
USA maintainer@bioconductor.org

Changes in R

From version 3.2.4 to version 3.3.1 patched

by the R Core Team

CHANGES IN R 3.3.1 patched

NEW FEATURES

- `extSoftVersion()` now reports the version (if any) of the `readline` library in use.
- Convenience function `hasName()` has been added; it is intended to replace the common idiom `!is.null(x$name)` without the usually unintended partial name matching.
- The version of LAPACK included in the sources has been updated to 3.6.1, a bug-fix release including a speedup for the non-symmetric case of `eigen()`.
- Use `options(deparse.max.lines)` to limit the number of lines recorded in .Traceback and other deparsing activities.

INSTALLATION and INCLUDED SOFTWARE

- Versions of the `readline` library >= 6.3 had been changed so that terminal window resizes were not signalled to `readline`: code has been added using a explicit signal handler to work around that (when R is compiled against readline >= 6.3). (PR#16604)
- `configure` works better with Oracle Developer Studio 12.5.

UTILITIES

- `R CMD check` reports more dubious flags in files 'src/Makevars[.in]', including '`-w`' and '`-g`'.
- `R CMD check` has been set up to filter important warnings from recent versions of `gfortran` with '`-Wall -pedantic`': this now reports non-portable GNU extensions such as out-of-order declarations.

BUG FIXES

- The check for non-portable flags in `R CMD check` could be stymied by 'src/Makevars' files which contained targets.
- (Windows only) When using certain desktop themes in Windows 7 or higher, `Alt-Tab` could cause `Rterm` to stop accepting input. (PR#14406; patch submitted by Jan Gleixner.)
- `pretty(d,..)` behaves better for date-time `d` (PR#16923).
- When a class name matches multiple classes in the cache, perform a dynamic search in order to obey namespace imports. This should eliminate annoying messages about multiple hits in the class cache. Also, pass along the package from the `ClassExtends` object when looking up superclasses in the cache.
- `sample(NA_real_)` now works.
- Packages using non-ASCII encodings in their code did not install data properly on systems using different encodings.

- `merge(df1,df2)` now also works for data frames with column names `"na.last"`, `"decreasing"`, or `"method"`. (PR#17119)
- `contour()` caused a segfault if the `labels` argument had length zero. (Reported by Bill Dunlap.)
- `unique(warnings())` works more correctly, thanks to a new `duplicated.warnings()` method.
- `findInterval(x,vec = numeric(),all.inside = TRUE)` now returns `0`s as documented. (Reported by Bill Dunlap.)
- (Windows only) `R CMD SHLIB` failed when a symbol in the resulting library had the same name as a keyword in the '.def' file. (PR#17130)
- `pmax()` and `pmin()` now work with (more ?) classed objects, such as `"Matrix"` from the **Matrix** package, as documented for a long time.
- `axis(side,x = D)` and hence `Axis()` and `plot()` now work correctly for `"Date"` and time objects `D`, even when "time goes backward", e.g., with decreasing `xlim`. (Reported by William May).

CHANGES IN R 3.3.1

BUG FIXES

- `R CMD INSTALL` and hence `install.packages()` gave an internal error installing a package called **description** from a tarball on a case-insensitive file system.
- `match(x,t)` (and hence `x %in% t`) failed when `x` was of length one, and either `character` and `x` and `t` only differed in their `Encoding` or when `x` and `t` where `complex` with `NA`s or `NaN`s. (PR#16885.)
- `unloadNamespace(ns)` also works again when `ns` is a 'namespace', as from `getNamespace()`.
- `rgamma(1,Inf)` or `rgamma(1,0,0)` no longer give `NaN` but the correct limit.
- `length(baseenv())` is correct now.
- `pretty(d,..)` for date-time `d` rarely failed when `"halfmonth"` time steps were tried (PR#16923) and on 'inaccurate' platforms such as 32-bit Windows or a configuration with `--disable-long-double`; see comment #15 of PR#16761.
- In `text.default(x,y,labels)`, the rarely(?) used default for `labels` is now correct also for the case of a 2-column matrix `x` and missing `y`.
- `as.factor(c(a = 1L))` preserves `names()` again as in R < 3.1.0.
- `strtrim(""[0],0[0])` now works.
- Use of `Ctrl-C` to terminate a reverse incremental search started by `Ctrl-R` in the `readline`-based Unix terminal interface is now supported when R was compiled against `readline` >= 6.0 (`Ctrl-G` always worked). (PR#16603)
- `diff(<difftime>)` now keeps the `"units"` attribute, as subtraction already did, PR#16940.

CHANGES IN R 3.3.0

SIGNIFICANT USER-VISIBLE CHANGES

- `nchar(x,*)`'s argument `keepNA` governing how the result for `NA`s in `x` is determined, gets a new default `keepNA = NA` which returns `NA` where `x` is `NA`, except for `type = "width"` which still returns 2, the formatting / printing width of `NA`.
- All builds have support for '`https:`' URLs in the default methods for `download.file()`, `url()` and code making use of them.

  Unfortunately that cannot guarantee that any particular '`https:`' URL can be accessed. For example, server and client have to successfully negotiate a cryptographic protocol (TLS/SSL, ...) and the server's identity has to be verifiable *via* the available certificates. Different access methods may allow different protocols or use private certificate bundles: we encountered a '`https:`' CRAN mirror which could be accessed by one browser but not by another nor by `download.file()` on the same Linux machine.

NEW FEATURES

- The `print` method for `methods()` gains a `byclass` argument.
- New functions `validEnc()` and `validUTF8()` to give access to the validity checks for inputs used by `grep()` and friends.
- Experimental new functionality for S3 method checking, notably `isS3method()`.

  Also, the names of the R 'language elements' are exported as character vector `tools::langElts`.
- `str(x)` now displays `"Time-Series"` also for matrix (multivariate) time-series, i.e. when `is.ts(x)` is true.
- (Windows only) The GUI menu item to install local packages now accepts '*.tar.gz' files as well as '*.zip' files (but defaults to the latter).
- New programmeR's utility function `chkDots()`.
- `D()` now signals an error when given invalid input, rather than silently returning `NA`. (Request of John Nash.)
- `formula` objects are slightly more "first class": e.g., `formula()` or `new("formula",y ~ x)` are now valid. Similarly, for `"table"`, `"ordered"` and `"summary.table"`. Packages defining S4 classes with the above S3/S4 classes as slots should be reinstalled.
- New function `strrep()` for repeating the elements of a character vector.
- `rapply()` preserves attributes on the list when `how = "replace"`.
- New S3 generic function `sigma()` with methods for extracting the estimated standard deviation aka "residual standard deviation" from a fitted model.
- `news()` now displays R and package news files within the HTML help system if it is available. If no news file is found, a visible `NULL` is returned to the console.
- `as.raster(x)` now also accepts `raw` arrays `x` assuming values in `0:255`.
- Subscripting of matrix/array objects of type `"expression"` is now supported.
- `type.convert("i")` now returns a factor instead of a complex value with zero real part and missing imaginary part.

- Graphics devices `cairo_pdf()` and `cairo_ps()` now allow non-default values of the cairographics 'fallback resolution' to be set.

  This now defaults to 300 on all platforms: that is the default documented by cairographics, but apparently was not used by all system installations.

- `file()` gains an explicit `method` argument rather than implicitly using `getOption("url.method","default")`.

- Thanks to a patch from Tomas Kalibera, `x[x != 0]` is now typically faster than `x[which(x != 0)]` (in the case where `x` has no NAs, the two are equivalent).

- `read.table()` now always uses the names for a named `colClasses` argument (previously names were only used when `colClasses` was too short). (In part, wish of PR#16478.)

- (Windows only) `download.file()` with default `method = "auto"` and a 'ftps://' URL chooses `"libcurl"` if that is available.

- The out-of-the box Bioconductor mirror has been changed to one using 'https://': use `chooseBioCmirror()` to choose a 'http://' mirror if required.

- The data frame and formula methods for `aggregate()` gain a `drop` argument.

- `available.packages()` gains a `repos` argument.

- The undocumented switching of methods for `url()` on 'https:' and 'ftps:' URLs is confined to `method = "default"` (and documented).

- `smoothScatter()` gains a `ret.selection` argument.

- `qr()` no longer has a `...` argument to pass additional arguments to methods.

- `[` has a method for class `"table"`.

- It is now possible (again) to `replayPlot()` a display list snapshot that was created by `recordPlot()` in a different R session.

  It is still not a good idea to use snapshots as a persistent storage format for R plots, but it is now not completely silly to use a snapshot as a format for transferring an R plot between two R sessions.

  The underlying changes mean that packages providing graphics devices (e.g., **Cairo**, **RSvgDevice**, **cairoDevice**, **tikzDevice**) will need to be reinstalled.

  Code for restoring snapshots was contributed by Jeroen Ooms and JJ Allaire.

  Some testing code is available at https://github.com/pmur002/R-display-list.

- `tools::undoc(dir = D)` and `codoc(dir = D)` now also work when `D` is a directory whose `normalizePath()`ed version does not end in the package name, e.g. from a symlink.

- `abbreviate()` has more support for multi-byte character sets – it no longer removes bytes within characters and knows about Latin vowels with accents. It is still only really suitable for (most) European languages, and still warns on non-ASCII input.

  `abbreviate(use.classes = FALSE)` is now implemented, and that is more suitable for non-European languages.

- `match(x,table)` is faster (sometimes by an order of magnitude) when `x` is of length one and `incomparables` is unchanged, thanks to Peter Haverty (PR#16491).

- More consistent, partly not back-compatible behavior of `NA` and `NaN` coercion to complex numbers, operations less often resulting in complex `NA` (`NA_complex_`).

- `lengths()` considers methods for `length` and `[[` on `x`, so it should work automatically on any objects for which appropriate methods on those generics are defined.
- The logic for selecting the default screen device on OS X has been simplified: it is now `quartz()` if that is available even if environment variable `DISPLAY` has been set by the user.

  The choice can easily be overridden *via* environment variable `R_INTERACTIVE_DEVICE`.
- On Unix-like platforms which support the `getline` C library function, `system(*,intern = TRUE)` no longer truncates (output) lines longer than 8192 characters, thanks to Karl Millar. (PR#16544)
- `rank()` gains a `ties.method = "last"` option, for convenience (and symmetry).
- `regmatches(invert = NA)` can now be used to extract both non-matched and matched substrings.
- `data.frame()` gains argument `fix.empty.names`; `as.data.frame.list()` gets new `cut.names`, `col.names` and `fix.empty.names`.
- `plot(x ~ x,*)` now warns that it is the same as `plot(x ~ 1,*)`.
- `recordPlot()` has new arguments `load` and `attach` to allow package names to be stored as part of a recorded plot. `replayPlot()` has new argument `reloadPkgs` to load/attach any package names that were stored as part of a recorded plot.
- S4 dispatch works within calls to `.Internal()`. This means explicit S4 generics are no longer needed for `unlist()` and `as.vector()`.
- Only font family names starting with '`"Hershey"`' (and not '`"Her"`' as before) are given special treatment by the graphics engine.
- S4 values are automatically coerced to vector (via `as.vector`) when subassigned into atomic vectors.
- `findInterval()` gets a `left.open` option.
- The version of LAPACK included in the sources has been updated to 3.6.0, including those 'deprecated' routines which were previously included. *Ca* 40 double-complex routines have been added at the request of a package maintainer.

  As before, the details of what is included are in 'src/modules/lapack/README' and this now gives information on earlier additions.
- `tapply()` has been made considerably more efficient without changing functionality, thanks to proposals from Peter Haverty and Suharto Anggono. (PR#16640)
- `match.arg(arg)` (the one-argument case) is faster; so is `sort.int()`. (PR#16640)
- The `format` method for `object_size` objects now also accepts "binary" units such as `"KiB"` and e.g., `"Tb"`. (Partly from PR#16649.)
- Profiling now records calls of the form `foo::bar` and some similar cases directly rather than as calls to `<Anonymous>`. Contributed by Winston Chang.
- New string utilities `startsWith(x,prefix)` and `endsWith(x,suffix)`. Also provide speedups for some `grepl("^...",*)` uses (related to proposals in PR#16490).
- Reference class finalizers run at exit, as well as on garbage collection.
- Avoid **parallel** dependency on **stats** for port choice and random number seeds. (PR#16668)

- The radix sort algorithm and implementation from **data.table** (`forder`) replaces the previous radix (counting) sort and adds a new method for `order()`. Contributed by Matt Dowle and Arun Srinivasan, the new algorithm supports logical, integer (even with large values), real, and character vectors. It outperforms all other methods, but there are some caveats (see `?sort`).

- The `order()` function gains a `method` argument for choosing between `"shell"` and `"radix"`.

- New function `grouping()` returns a permutation that stably rearranges data so that identical values are adjacent. The return value includes extra partitioning information on the groups. The implementation came included with the new radix sort.

- `rhyper(nn,m,n,k)` no longer returns `NA` when one of the three parameters exceeds the maximal integer.

- `switch()` now warns when no alternatives are provided.

- `parallel::detectCores()` now has default `logical = TRUE` on all platforms – as this was the default on Windows, this change only affects Sparc Solaris.

  Option `logical = FALSE` is now supported on Linux and recent versions of OS X (for the latter, thanks to a suggestion of Kyaw Sint).

- `hist()` for `"Date"` or `"POSIXt"` objects would sometimes give misleading labels on the breaks, as they were set to the day before the start of the period being displayed. The display format has been changed, and the shift of the start day has been made conditional on `right = TRUE` (the default). (PR#16679)

- R now uses a new version of the logo (donated to the R Foundation by RStudio). It is defined in '.svg' format, so will resize without unnecessary degradation when displayed on HTML pages—there is also a vector PDF version. Thanks to Dirk Eddelbuettel for producing the corresponding X11 icon.

- New function `.traceback()` returns the stack trace which `traceback()` prints.

- `lengths()` dispatches internally.

- `dotchart()` gains a `pt.cex` argument to control the size of points separately from the size of plot labels. Thanks to Michael Friendly and Milan Bouchet-Valat for ideas and patches.

- `as.roman(ch)` now correctly deals with more diverse character vectors `ch`; also arithmetic with the resulting roman numbers works in more cases. (PR#16779)

- `prcomp()` gains a new option `rank.` allowing to directly aim for less than `min(n,p)` PC's. The `summary()` and its `print()` method have been amended, notably for this case.

- `gzcon()` gains a new option `text`, which marks the connection as text-oriented (so e.g. `pushBack()` works). It is still always opened in binary mode.

- The `import()` namespace directive now accepts an argument `except` which names symbols to exclude from the imports. The `except` expression should evaluate to a character vector (after substituting symbols for strings). See Writing R Extensions.

- New convenience function `Rcmd()` in package **tools** for invoking `R CMD` tools from within R.

- New functions `makevars_user()` and `makevars_site()` in package **tools** to determine the location of the user and site specific 'Makevars' files for customizing package compilation.

UTILITIES

- `R CMD check` has a new option '`--ignore-vignettes`' for use with non-Sweave vignettes whose '`VignetteBuilder`' package is not available.
- `R CMD check` now by default checks code usage (*via* **codetools**) with only the base package attached. Functions from default packages other than **base** which are used in the package code but not imported are reported as undefined globals, with a suggested addition to the `NAMESPACE` file.
- `R CMD check --as-cran` now also checks DOIs in package 'CITATION' and Rd files.
- `R CMD Rdconv` and `R CMD Rd2pdf` each have a new option '`--RdMacros=pkglist`' which allows Rd macros to be specified before processing.

DEPRECATED AND DEFUNCT

- The previously included versions of `zlib`, `bzip2`, `xz` and PCRE have been removed, so suitable external (usually system) versions are required (see the 'R Installation and Administration' manual).
- The unexported and undocumented Windows-only devices `cairo_bmp()`, `cairo_png()` and `cairo_tiff()` have been removed. (These devices should be used as e.g. `bmp(type = "cairo")`.)
- (Windows only) Function `setInternet2()` has no effect and will be removed in due course. The choice between methods `"internal"` and `"wininet"` is now made by the `method` arguments of `url()` and `download.file()` and their defaults can be set *via* options. The out-of-the-box default remains `"wininet"` (as it has been since R 3.2.2).
- `[<-` with an S4 value into a list currently embeds the S4 object into its own list such that the end result is roughly equivalent to using `[[<-`. That behavior is deprecated. In the future, the S4 value will be coerced to a list with `as.list()`.
- Package **tools**' functions `package.dependencies()`, `pkgDepends()`, etc are deprecated now, mostly in favor of `package_dependencies()` which is both more flexible and efficient.

INSTALLATION and INCLUDED SOFTWARE

- Support for very old versions of `valgrind` (e.g., 3.3.0) has been removed.
- The included `libtool` script (generated by `configure`) has been updated to version 2.4.6 (from 2.2.6a).
- `libcurl` version 7.28.0 or later with support for the `https` protocol is required for installation (except on Windows).
- BSD networking is now required (except on Windows) and so `capabilities("http/ftp")` is always true.
- `configure` uses `pkg-config` for PNG, TIFF and JPEG where this is available. This should work better with multiple installs and with those using static libraries.
- The minimum supported version of OS X is 10.6 ('Snow Leopard'): even that has been unsupported by Apple since 2012.
- The `configure` default on OS X is '`--disable-R-framework`': enable this if you intend to install under '/Library/Frameworks' and use with `R.app`.

- The minimum preferred version of PCRE has since R 3.0.0 been 8.32 (released in Nov 2012). Versions 8.10 to 8.31 are now deprecated (with warnings from `configure`), but will still be accepted until R 3.4.0.
- `configure` looks for C functions `__cospi`, `__sinpi` and `__tanpi` and uses these if `cospi` *etc* are not found. (OS X is the main instance.)
- (Windows) R is now built using gcc 4.9.3. This build will require recompilation of at least those packages that include C++ code, and possibly others. A build of R-devel using the older toolchain will be temporarily available for comparison purposes.

  During the transition, the environment variable `R_COMPILED_BY` has been defined to indicate which toolchain was used to compile R (and hence, which should be used to compile code in packages). The `COMPILED_BY` variable described below will be a permanent replacement for this.
- (Windows) A `make` and `R CMD config` variable named `COMPILED_BY` has been added. This indicates which toolchain was used to compile R (and hence, which should be used to compile code in packages).

PACKAGE INSTALLATION

- The `make` macro `AWK` which used to be made available to files such as 'src/Makefile' is no longer set.

C-LEVEL FACILITIES

- The API call `logspace_sum` introduced in R 3.2.0 is now remapped as an entry point to `Rf_logspace_sum`, and its first argument has gained a `const` qualifier. (PR#16470)

  Code using it will need to be reinstalled.

  Similarly, entry point `log1pexp` also defined in 'Rmath.h' is remapped there to `Rf_log1pexp`
- `R_GE_version` has been increased to 11.
- New API call `R_orderVector1`, a faster one-argument version of `R_orderVector`.
- When R headers such as 'R.h' and 'Rmath.h' are called from C++ code in packages they include the C++ versions of system headers such as '<cmath>' rather than the legacy headers such as '<math.h>'. (Headers 'Rinternals.h' and 'Rinterface.h' already did, and inclusion of system headers can still be circumvented by defining `NO_C_HEADERS`, including as from this version for those two headers.)

  The manual has long said that R headers should **not** be included within an `extern "C"` block, and almost all the packages affected by this change were doing so.
- Including header 'S.h' from C++ code would fail on some platforms, and so gives a compilation error on all.
- The deprecated header 'Rdefines.h' is now compatible with defining `R_NO_REMAP`.
- The connections API now includes a function `R_GetConnection()` which allows packages implementing connections to convert R `connection` objects to `Rconnection` handles used in the API. Code which previously used the low-level R-internal `getConnection()` entry point should switch to the official API.

BUG FIXES

- C-level `asChar(x)` is fixed for when `x` is not a vector, and it returns `"TRUE"/"FALSE"` instead of `"T"/"F"` for logical vectors.
- The first arguments of `.colSums()` etc (with an initial dot) are now named `x` rather than `X` (matching `colSums()`): thus error messages are corrected.
- A `coef()` method for class `"maov"` has been added to allow `vcov()` to work with multivariate results. (PR#16380)
- `method = "libcurl"` connections signal errors rather than retrieving HTTP error pages (where the ISP reports the error).
- `xpdrows.data.frame()` was not checking for unique row names; in particular, this affected assignment to non-existing rows via numerical indexing. (PR#16570)
- `tail.matrix()` did not work for zero rows matrices, and could produce row "labels" such as `"[1e+05,]"`.
- Data frames with a column named `"stringsAsFactors"` now format and print correctly. (PR#16580)
- `cor()` is now guaranteed to return a value with absolute value less than or equal to 1. (PR#16638)
- Array subsetting now keeps `names(dim(.))`.
- Blocking socket connection selection recovers more gracefully on signal interrupts.
- The `data.frame` method of `rbind()` construction `row.names` works better in borderline integer cases, but may change the names assigned. (PR#16666)
- (X11 only) `getGraphicsEvent()` miscoded buttons and missed mouse motion events. (PR#16700)
- `methods(round)` now also lists `round.POSIXt`.
- `tar()` now works with the default `files = NULL`. (PR#16716)
- Jumps to outer contexts, for example in error recovery, now make intermediate jumps to contexts where `on.exit()` actions are established instead of trying to run all `on.exit()` actions before jumping to the final target. This unwinds the stack gradually, releases resources held on the stack, and significantly reduces the chance of a segfault when running out of C stack space. Error handlers established using `withCallingHandlers()` and `options("error")` specifications are ignored when handling a C stack overflow error as attempting one of these would trigger a cascade of C stack overflow errors. (These changes resolve PR#16753.)
- The spacing could be wrong when printing a complex array. (Report and patch by Lukas Stadler.)
- `pretty(d,n,min.n,*)` for date-time objects `d` works again in border cases with large `min.n`, returns a `labels` attribute also for small-range dates and in such cases its returned length is closer to the desired `n`. (PR#16761) Additionally, it finally does cover the range of `d`, as it always claimed.
- `tsp(x) <-NULL` did not handle correctly objects inheriting from both `"ts"` and `"mts"`. (PR#16769)
- `install.packages()` could give false errors when `options("pkgType")` was `"binary"`. (Reported by Jose Claudio Faria.)

- A bug fix in R 3.0.2 fixed problems with `locator()` in X11, but introduced problems in Windows. Now both should be fixed. (PR#15700)
- `download.file()` with `method = "wininet"` incorrectly warned of download file length difference when reported length was unknown. (PR#16805)
- `diag(NULL,1)` crashed because of missed type checking. (PR#16853)