Time Series Forecasting with KNN in R: the tsfknn Package

by Francisco Martínez, María P. Frías, Francisco Charte and Antonio J. Rivera

Abstract In this paper the **tsfknn** package for time series forecasting using *k*-nearest neighbor regression is described. This package allows users to specify a KNN model and to generate its forecasts. The user can choose among different multi-step ahead strategies and among different functions to aggregate the targets of the nearest neighbors. It is also possible to assess the forecast accuracy of the KNN model.

Introduction

Time series forecasting has been performed traditionally using statistical methods such as ARIMA models or exponential smoothing (Hyndman and Athanasopoulos, 2014). However, the last decades have witnessed the use of computational intelligence techniques to forecast time series. Although artificial neural networks is the most prominent machine learning technique used in time series forecasting (Zhang et al., 1998), other approaches, such as Gaussian Processes (Andrawis et al., 2011) or KNN (Martínez et al., 2017), have also been applied. Compared with classical statistical models, computational intelligence methods exhibit interesting features, such as their nonlinearity or the lack of an underlying model, that is, they are non-parametric.

Statistical methodologies for time series forecasting are present in R as excellent packages. For example, the **forecast** package (Hyndman and Khandakar, 2008) includes implementations of ARIMA, exponential smoothing, the Theta method (Hyndman and Billah, 2003) or basic techniques that can be used as benchmark methods—such as the random walk approach. On the other hand, although a great variety of computational intelligence approaches for regression are available in R, such us the **caret** package (Kuhn, 2008), these approaches cannot be directly applied to time series forecasting. Fortunately, some new packages are filling this gap. For example, the **nnfor** package (Kourentzes, 2017) or the `nnetar` function from the **forecast** package allow users to predict time series using artificial neural networks.

KNN is a very popular algorithm used in classification and regression (Wu et al., 2007). This algorithm simply stores a collection of examples. In regression, each example consists of a vector of features describing the example and its associated numeric target value. Given a new example, KNN finds its k most similar examples, called nearest neighbors, according to a distance metric such as the Euclidean distance, and predicts its value as an aggregation of the target values associated with its nearest neighbors. In this paper we describe the **tsfknn** R package for univariate time series forecasting using KNN regression.

The rest of the paper is organized as follows. Firstly, we explain how KNN regression can be applied in a time series forecasting context using the **tsfknn** package. Next, the different multi-step ahead strategies implemented in our package are explained. Some additional features of the package related to how the KNN model is specified are also discussed. The last sections explain how to assess the forecast accuracy of a model and compare the package with other R packages based on machine learning approaches.

Time series forecasting with KNN regression

In this section we first explain how KNN regression can be applied to forecast time series. Next, we describe how the **tsfknn** package can be used to forecast a time series.

As described above, KNN regression simply holds a collection of training instances. The i-th training instance consists of a vector of n features: $(f_1^i, f_2^i, \ldots, f_n^i)$, describing the instance and an associated target vector of m attributes: $(t_1^i, t_2^i, \ldots, t_m^i)$. Given a new instance, whose features are known—$(q_1, q_2, \ldots, q_n)$—but whose target is unknown, the features of the new instance are used to find its k most similar training instances according to the vectors of features and a similarity or distance metric. For example, assuming that the similarity metric is the Euclidean distance, the distance between the new instance and the i-th training instance is computed as follows:

$$\sqrt{\sum_{x=1}^{n} \left(f_x^i - q_x\right)^2} \tag{1}$$

The k training instances that are closest to the new instance are considered their k most similar instances or k nearest neighbors. KNN is based on learning by analogy. Given a new instance, we think that the targets of its nearest neighbors are probably similar to its unknown target. This way, the targets of the nearest neighbors are aggregated to predict the target of the new instance. For example, assuming that the targets or the k nearest neighbors are the vectors: $t^1, t^2, ..., t^k$, they can be averaged to predict the target of the new instance as:

$$\sum_{i=1}^{k} \frac{t^i}{k} \tag{2}$$

In short, KNN stores a collection of training instances described by n features. Each training instance represents a point in an n-dimensional space. Given a new instance, KNN finds its k closest instances in the n-dimensional space in the hope that their targets are similar to its unknown target.

Now, let us see how KNN can be applied to time series forecasting. In this case, the target associated with a training instance is a collection of values of the time series and the features describing the instance are lagged values of the target—that is, we have an autoregressive model. For example, let us start with a monthly time series containing 132 observations, i.e., 11 years: $t = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, ..., x_{132}\}$ and assume that we want to predict its next future month. Because we are going to predict only one value the target of a training instance is a value of the time series. Let us also assume that we decide that the features describing a target are its first twelve lagged values in the time series—that we will denote as lags 1:12. Therefore, the training instances or examples associated with the time series t are shown in Table 1. Now, let us see which is the new instance used to predict the next future value of the time series. Because we are using lags 1 to 12 as feature vector, the feature vector associated with the next future point is vector $(x_{121}, x_{122}, ..., x_{132})$, which is compounded of the last twelve values of the time series. If, for example, k is equal to 2 the 2-nearest neighbors of the new instance are found and their targets will be aggregated to predict the next future month. The rationale behind the use of KNN for time series forecasting is that a time series can contain repetitive patterns. Given the last pattern of a time series we look for similar patterns in the past in the hope that their subsequent patterns will be similar to the future values of the time series.

Table 1: Training examples for time series $t = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, ..., x_{132}\}$ for one-step ahead forecasting and lags 1:12 as feature vector

Features	Target
$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}$	x_{13}
$x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}$	x_{14}
$x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}$	x_{15}
...	...
$x_{120}, x_{121}, x_{122}, x_{123}, x_{124}, x_{125}, x_{126}, x_{127}, x_{128}, x_{129}, x_{130}, x_{131}$	x_{132}

Let us see now how the **tsfknn** package can be used to forecast a time series. In our package, you can consult the training examples associated with a KNN model with the `knn_examples` function:

```
> timeS <- window(nottem, end = c(1930, 12))
> pred <- knn_forecasting(timeS, h = 1, lags = 1:12, k = 2)
> head(knn_examples(pred))
     Lag12 Lag11 Lag10 Lag9 Lag8 Lag7 Lag6 Lag5 Lag4 Lag3 Lag2 Lag1   H1
[1,]  40.6  40.8  44.4 46.7 54.1 58.5 57.7 56.4 54.3 50.5 42.9 39.8 44.2
[2,]  40.8  44.4  46.7 54.1 58.5 57.7 56.4 54.3 50.5 42.9 39.8 44.2 39.8
[3,]  44.4  46.7  54.1 58.5 57.7 56.4 54.3 50.5 42.9 39.8 44.2 39.8 45.1
[4,]  46.7  54.1  58.5 57.7 56.4 54.3 50.5 42.9 39.8 44.2 39.8 45.1 47.0
[5,]  54.1  58.5  57.7 56.4 54.3 50.5 42.9 39.8 44.2 39.8 45.1 47.0 54.1
[6,]  58.5  57.7  56.4 54.3 50.5 42.9 39.8 44.2 39.8 45.1 47.0 54.1 58.7
```

Before consulting the training examples with `knn_examples`, you have to build the model. This is done with the function `knn_forecasting` that builds a model associated with a time series and uses the model to predict the future values of the time series. Let us see the main arguments of this function:

`timeS` : the time series to be forecast.

`h` : the forecasting horizon, that is, the number of future values to be predicted.

`lags` : an integer vector indicating the lagged values of the target used as its describing features in the examples—for instance, 1:12 means that lagged values 1 to 12 should be used.

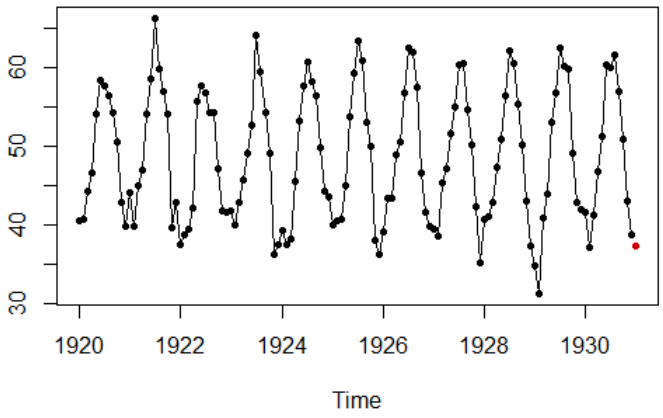

Figure 1: Result of plotting a prediction.

`k` : the number of nearest neighbors used by the KNN model.

`knn_forecasting` is very handy because, as commented above, it builds the KNN model and then uses the model to predict the time series. This function returns a `"knnForecast"` object with information of the model and its prediction. As we have seen above, you can use the function `knn_examples` to see the training examples associated with the model. You can also consult the prediction or plot it through the `"knnForecast"` object:

```
> pred$prediction
      Jan
1931 37.4
> plot(pred)
```

Figure 1 shows the result of plotting the prediction. It is also possible to see how the prediction was made. That is, you can consult the new instance whose target was predicted and its nearest neighbors. This information is obtained with the `nearest_neighbors` function applied to a `"knnForecast"` object:

```
> nearest_neighbors(pred)
$`instance`
Lag 12 Lag 11 Lag 10  Lag 9  Lag 8  Lag 7  Lag 6  Lag 5  Lag 4  Lag 3  Lag 2  Lag 1
  41.6   37.1   41.2   46.9   51.2   60.4   60.1   61.6   57.0   50.9   43.0   38.8

$nneighbors
  Lag 12 Lag 11 Lag 10 Lag 9 Lag 8 Lag 7 Lag 6 Lag 5 Lag 4 Lag 3 Lag 2 Lag 1   H1
1   40.8   41.1   42.8  47.3  50.9  56.4  62.2  60.5  55.4  50.2  43.0  37.3 34.8
2   39.3   37.5   38.3  45.5  53.2  57.7  60.8  58.2  56.4  49.8  44.4  43.6 40.0
```

Because we have used lags 1:12 as features, the features associated with the next future value of the time series are the last twelve values of the time series. The targets of the two most similar examples or nearest neighbors are 34.8 and 40. Their average is the prediction: 37.4. A nice plot including the new instance, its nearest neighbors and the prediction can be obtained as follows:

```
> library(ggplot2)
> autoplot(pred, highlight = "neighbors", faceting = FALSE)
```

The result of executing this code snippet is shown in Figure 2. To recapitulate, in order to specify a KNN model you have to set:

- The lags used to build the KNN examples. They determine the lagged values used as features or autoregressive explanatory variables.
- k, that is, the number of nearest neighbors used in the prediction.
- The forecasting horizon, that is, the number of future points to predict.

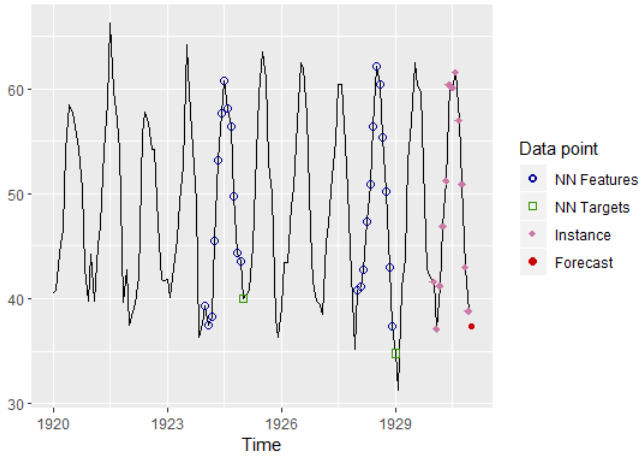

Figure 2: Plotting the instance, its nearest neighbors and the prediction.

Multi-step ahead strategies

In the previous section we have seen an example of one-step ahead prediction with KNN. Nonetheless, it is very common to forecast more than one value into the future. To this end, a multi-step ahead strategy has to be chosen (Ben Taieb et al., 2012). Our package implements two common strategies: the MIMO—Multiple Input Multiple Output—approach and the recursive or iterative approach. It must be noted that when only one future value is predicted both strategies are equivalent. In the next subsections these strategies are explained, together with examples of how they can be used in our package.

The MIMO strategy

The Multiple Input Multiple Output strategy is commonly applied with KNN and it is characterized by the use of a vector of target values. The length of this vector is equal to the number of periods to be forecast. For example, let us assume that we are working with a time series giving the monthly totals of car drivers killed in Great Britain and we want to forecast the number of deaths for the next 12 months. In this situation, a good choice for the lags used as features would be 1:12, i.e., the totals of car drivers killed in the previous 12 months—an explanation about why lags 1:12 are a good choice is given in the section about default parameters. If the MIMO strategy is chosen, then a training example consists of:

- a target vector with the number of deaths of 12 consecutive months and
- a feature vector with the number of deaths in the previous 12 consecutive months—just before the 12 months of the target vector.

The new instance would be the number of car drivers killed in the last 12 months of the time series. This way, we would look for the number of deaths most similar to the last 12 months in the time series and we would predict an aggregation of their subsequent 12 months. In the following example we predict the next 12 months using the MIMO strategy:

```
> timeS <- window(UKDriverDeaths, end = c(1979, 12))
> pred <- knn_forecasting(timeS, h = 12, lags = 1:12, k = 2, msas = "MIMO")
> autoplot(pred, highlight = "neighbors", faceting = FALSE)
```

The forecast for the next 12 months can be seen in Figure 3. The last 12 values of the time series are the features of the new instance whose target has to be predicted. The two sequences of 12 consecutive values most similar to this instance are found—in blue—and their subsequent 12 values—in green—are averaged to obtain the prediction—in red.

The recursive strategy

The recursive or iterative strategy is the approach used by ARIMA or exponential smoothing to forecast several periods. In this strategy a model that only forecasts one-step ahead is used. Therefore,

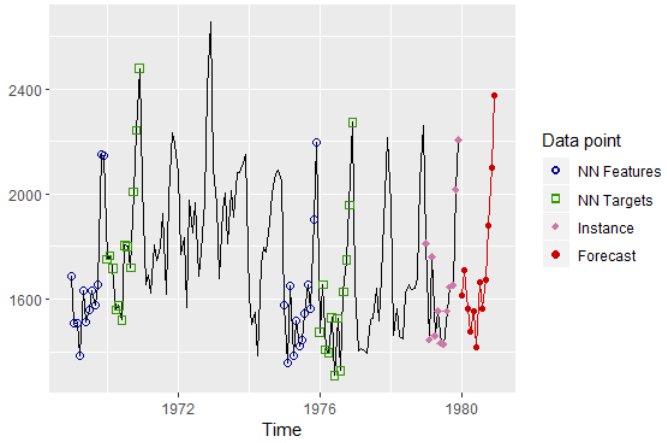

Figure 3: Applying the MIMO strategy to forecast the next 12 months.

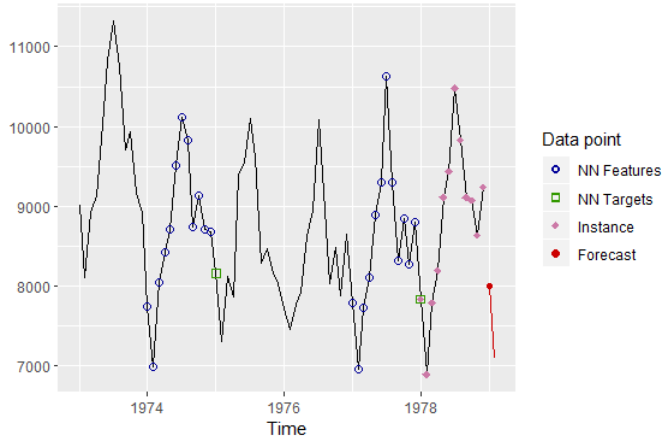

Figure 4: Applying the recursive strategy to forecast horizon 1.

the model is applied iteratively to forecast all the future periods. When historical observations to be used as features of the new instance are unavailable, previous predictions are used instead.

Because the recursive strategy uses a one-step ahead model, this means that, in the case of KNN, the target of a training example only contains one value. For instance, let us see how the recursive strategy works with the following example in which the next two future months of a monthly time series are predicted:

```
> pred <- knn_forecasting(USAccDeaths, h = 2, lags = 1:12, k = 2, msas = "recursive")
> autoplot(pred, highlight = "neighbors")
```

In this example we have used lags 1:12 to specify the features of an example. To predict the first future point the last 12 values of the time series are used as "its features"—see Figure 4. To predict the second future point "its features" are the last eleven values of the time series and the prediction for the first future point—see Figure 5.

Setting the KNN parameters

In this section several additional features of our package related to model selection are described. In order to select a KNN model the following parameters have to be chosen:

- The distance function used to find the nearest neighbors.
- The combination function used to aggregate the targets of the nearest neighbors.
- The number of nearest neighbors, that is, the k parameter.
- The lags used as autoregressive explanatory variables, that is, the features describing the training examples.

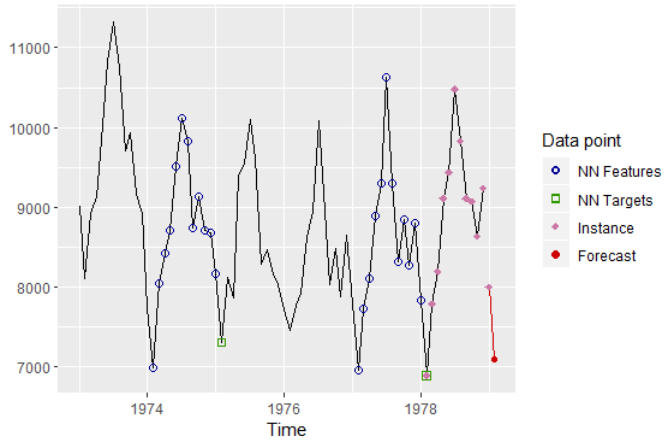

Figure 5: Applying the recursive strategy to forecast horizon 2: the previous prediction is used as feature.

- The multi-step ahead strategy.

In the following subsections some features related to setting the KNN parameters are explained.

Distance and combination function

Our package uses the Euclidean distance to find the nearest neighbors, although we can implement other distance metrics in the future.

Regarding the combination function, the targets of the k nearest neighbors are averaged by default. However, it is possible to combine the targets using other aggregation functions. Currently, our package allows users to choose among the mean, the median and a weighted combination. In order to select the combination function the `cb` parameter of the `knn_forecasting` function has to be used.

Next, we explain how the weighted combination is computed. The goal is to give more weight to the closer neighbors. Let us denote as d_i and t_i the distance between the i-th nearest neighbor and the the new instance and the target of the i-th nearest neighbor respectively. Then, we define $w_i = 1/d_i^2$ to be the reciprocal of the squared distance to the i-th nearest neighbor and the prediction is computed as the following weighted combination of the targets:

$$\frac{\sum_{i=1}^{k} w_i t_i}{\sum_{i=1}^{k} w_i} \tag{3}$$

This scheme fails when the distance to a training example is 0. In this special case, the target of this training example, whose features are identical to the new instance, is selected as the prediction.

An ensemble of several models with different k parameters

In order to specify a KNN model the k parameter has to be chosen. If k is too small, then the prediction can more easily be affected by noise. On the other hand, if k is too large, then we are using examples far apart from the new instance. Several strategies can be used to choose the k parameter. A first, fast, straightforward solution is to use some heuristic—it is recommended setting k to the square root of the number of training examples. Another approach is to select k using an optimization tool on a validation set. k should minimize a forecast accuracy measure such as MAPE (Hyndman and Koehler, 2006). It should be noted that the optimization strategy is very time-consuming.

A third strategy explored in (Martínez et al., 2017) is to use several KNN models with different k values. Each KNN model generates its forecasts and the forecasts of the different models are averaged to produce the final forecast. This strategy is based on the success of model combination in time series forecasting (Hibon and Evgeniou, 2005). In this way, the use of a time-consuming optimization tool is avoided and the forecasts are not based on a unique k value. In our package you can use this strategy specifying a vector of k values:

```
> pred <- knn_forecasting(ldeaths, h = 12, lags = 1:12, k = c(2, 4))
```

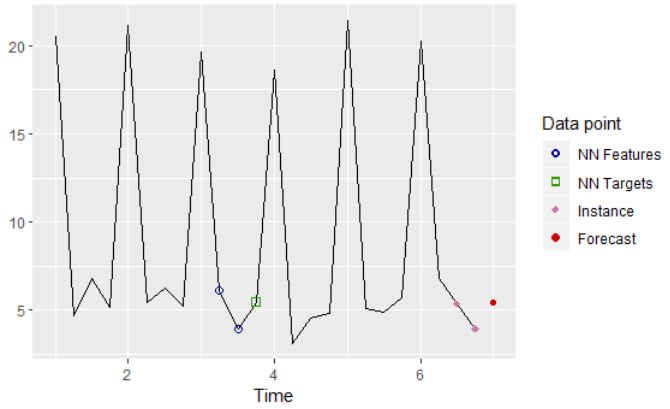

Figure 6: Quarterly time series with a strong seasonal pattern and lags 1:2.

```
> pred$prediction
          Jan      Feb      Mar      Apr      May      Jun      Jul      Aug      Sep
1980 2865.375 2866.250 2728.875 2189.000 1816.000 1625.875 1526.250 1404.250 1354.000
          Oct      Nov      Dec
1980 1541.250 1699.250 2198.750
```

In this example, two KNN models with k equal to 2 and 4 respectively are built. The forecast is the average of the forecasts produced by both models.

It also must be noted that if the targets are combined using weights, then the choice of the k parameter is less important because neighbors lose weight as they move away from the new instance. When the weighted combination is chosen, it even makes sense to set k to the number of training examples, as generalized regression neural networks do (Yan, 2012).

Default parameters

Sometimes a great number of time series have to be forecast. In that situation, an automatic way of generating the forecasts is very useful. In this sense, our package is able to use sensible default parameters. If the user only specifies the time series and the forecasting horizon the KNN parameters are selected as follows:

- As multi-step ahead strategy MIMO is chosen.
- The combination function used to aggregate the targets is the mean.
- k is selected as a combination of three models using 3, 5 and 7 nearest neighbors respectively.
- In order to select the autoregressive lags the following strategy is used. If the time series is seasonal and the length of the seasonal period is m, then lags 1:m are used. For example, for quarterly data lags 1:4 are selected and for monthly data lags 1:12. This way, seasonal patterns can be captured more easily. Let us see why—see also (Martínez et al., 2018). In Figure 6 an artificial quarterly time series with a strong seasonal pattern is shown. The first quarter has a mean level higher than the other quarters, which have a similar level. The autoregressive lags are lags 1:2 and we want to generate a one-step ahead forecast using only the nearest neighbor. These autoregressive lags can lead to an unsuitable forecast. As can be seen in the figure, the nearest neighbor has a fourth quarter as its target when a first quarter value is going to be predicted. In Figure 7 lags 1:4 are used and the target associated with the nearest neighbor is a first quarter value. If the time series is not seasonal, for example yearly data, then the lags with significant autocorrelation in the partial autocorrelation function (PACF) are selected. Although the PACF only tests for linear relationships, experience has shown us that this is an effective way of selecting input variables (Martínez et al., 2017). If none of the previous two conditions are met, then lags 1:5 are used. Notice that this way of selecting the autoregressive lags is quite fast.

Evaluating forecast accuracy

Once a model has been built, it is natural to want to assess its forecast accuracy. In the **tsfknn** package this is done with the `rolling_origin` function. This function uses the classical approach of dividing a

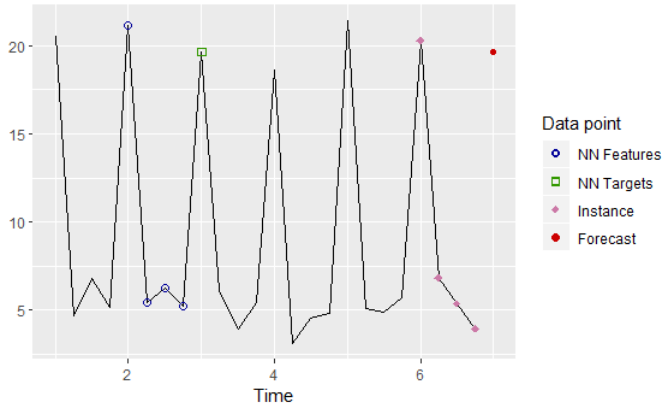

Figure 7: Quarterly time series with a strong seasonal pattern and lags 1:4.

dataset into training and test sets. In time series forecasting the test set consists of the last observations of the time series. For example:

```
> pred <- knn_forecasting(ldeaths, h = 12, lags = 1:12, k = 2)
> ro <- rolling_origin(pred, h = 6, rolling = FALSE)
```

As mentioned above, `knn_forecasting` builds a KNN model and returns a `"knnForecast"` object with information about the model. The `rolling_origin` function takes a `"knnForecast"` object as its first parameter. From this, information about the time series and the metaparameters of the KNN model is obtained; for example, the autoregressive lags, the number of nearest neighbors or the multi-step ahead strategy. The second parameter of `rolling_origin` is the size of the test set. In the example, the size is 6 and, therefore, the last 6 observations of the time series will be used as test set and the remaining observations as training set. `rolling_origin` returns a `"knnForecastRO"` object with information about the evaluation. For example, the test set, predictions and errors can be consulted:

```
> print(ro$test_sets)
      h=1  h=2  h=3  h=4  h=5  h=6
[1,] 1461 1354 1333 1492 1781 1915
> print(ro$predictions)
        h=1    h=2    h=3  h=4    h=5  h=6
[1,] 1513.5 1363.5 1351.5 1567 1587.5 2392
> print(ro$errors)
       h=1  h=2   h=3 h=4   h=5  h=6
[1,] -52.5 -9.5 -18.5 -75 193.5 -477
```

It is also possible to consult several forecasting accuracy measures about the predictions:

```
> ro$global_accu
      RMSE        MAE       MAPE
213.613748 137.666667   7.747168
```

The forecasting accuracy measures are: root mean square error, mean absolute error and mean absolute percentage error. A plot of the time series and the forecasts can be obtained:

```
> plot(ro)
```

The result of this plot can be seen in Figure 8. In this figure the last six observations of the time series are the test set and the forecasts are the red points.

Evaluation based on a rolling origin

A more sophisticated version of training/test sets is to use a rolling origin evaluation. The idea is as follows. The last n observations of a time series are used as test data. Then n evaluations are performed. In the first evaluation the last n observations are the test set and the previous ones the training set. A model is fitted with the training set, the predictions are generated and the errors observed. In the second evaluation the last $n-1$ observations are used as test set, and so on until the last evaluation in which the test set is the last value of the time series. Figure 9 illustrates the different training and test sets for a rolling origin evaluation of the last six values of a time series. As can be observed, the

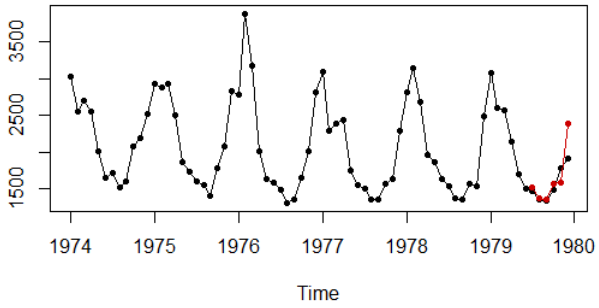

Figure 8: A time series and its forecast for the last 6 observations.

Figure 9: Training (blue) and test (red) sets for rolling origin evaluation with 6 observations.

origin of the forecasts rolls forward in time. The rolling origin technique aims to get the most out of the test data in terms of evaluation. For example, with n observations in the test data, n one-step ahead forecasts are generated, $n-1$ two-steps ahead forecasts, and so on. On the other hand, the rolling origin evaluation is a time-consuming task, because several models have to be fitted.

The function `rolling_origin` uses rolling origin evaluation by default:

```
> pred <- knn_forecasting(ldeaths, h = 12, lags = 1:12, k = 2)
> ro <- rolling_origin(pred, h = 6)
> print(ro$test_sets)
      h=1  h=2  h=3  h=4  h=5  h=6
[1,] 1461 1354 1333 1492 1781 1915
[2,] 1354 1333 1492 1781 1915   NA
[3,] 1333 1492 1781 1915   NA   NA
[4,] 1492 1781 1915   NA   NA   NA
[5,] 1781 1915   NA   NA   NA   NA
[6,] 1915   NA   NA   NA   NA   NA
> print(ro$predictions)
        h=1    h=2    h=3    h=4    h=5  h=6
[1,] 1513.5 1363.5 1351.5 1567.0 1587.5 2392
[2,] 1363.5 1351.5 1567.0 1587.5 2392.0   NA
[3,] 1351.5 1567.0 1587.5 2392.0     NA   NA
[4,] 1567.0 1587.5 2392.0     NA     NA   NA
[5,] 1587.5 2392.0     NA     NA     NA   NA
[6,] 2392.0     NA     NA     NA     NA   NA
> print(ro$errors)
        h=1    h=2    h=3    h=4    h=5  h=6
[1,]  -52.5   -9.5  -18.5  -75.0  193.5 -477
[2,]   -9.5  -18.5  -75.0  193.5 -477.0   NA
[3,]  -18.5  -75.0  193.5 -477.0     NA   NA
[4,]  -75.0  193.5 -477.0     NA     NA   NA
[5,]  193.5 -477.0     NA     NA     NA   NA
[6,] -477.0     NA     NA     NA     NA   NA
```

Each row in the output represents a different evaluation. Now, the `global_accu` field of the `"knnForecastRO"` object stores measures of the accuracy of all the forecasts in all the evaluations:

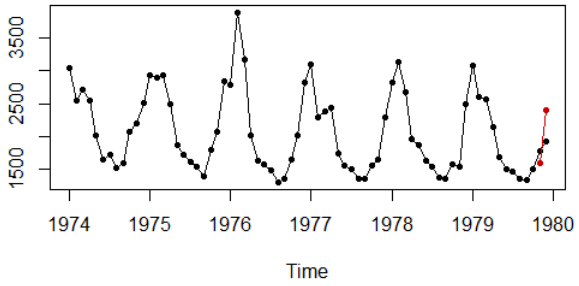

Figure 10: Forecast for the test set with the last two observations.

```
> ro$global_accu
     RMSE       MAE      MAPE
274.19569 202.69048  11.09727
```

The forecast accuracy measures for the different prediction horizons can also be consulted:

```
> ro$h_accu
            h=1         h=2       h=3       h=4       h=5       h=6
RMSE 213.613748 232.821283 260.25877 300.33107 363.98575 477.00000
MAE  137.666667 154.700000 191.00000 248.50000 335.25000 477.00000
MAPE   7.747168   8.577916  10.54699  13.60004  17.88665  24.90862
```

As expected, the errors grow with the increasing horizon. Finally, a plot of any of the different forecasts performed with the rolling origin evaluation can be obtained. For example:

```
> plot(ro, h = 2)
```

produces Figure 10.

A comparison with other time series forecasting packages

In this section our package is compared with other software for univariate time series forecasting in R. In the CRAN Task Views: Time Series Analysis, Econometrics and Finance some models for time series forecasting can be found, from GARCH models to ensembles of models, such us the **forecastHybrid** package (Shaub and Ellis, 2019). Most of the functions found in these packages use statistical models such as ARIMA or exponential smoothing. The **forecast** package is arguably the best package for time series forecasting. It implements all of the models that can be found in the **stats** package and many others such as the Theta method or multilayer perceptron. Furthermore, this package includes additional tools for plotting seasonal time series, doing Box-Cox transformations, etc.

The packages most closely related to our package are those that are based on machine learning approaches. Below, a brief description of these packages is given:

GMDH In this package the GMDH-type neural network algorithms are applied in order to perform short term forecasting for a univariate time series (Dag and Yozgatligil, 2016).

NTS This package contains a function, `NNsetting`, that allows users to create the examples needed to feed a neural network. However, the package does not allow forecasts to be generated directly.

tsDyn Allows users to predict a time series using a multi-layer perceptron with one hidden layer computed using the `nnet` function from the **nnet** package.

nnfor Uses the **neuralnet** package to build multi-layer perceptrons. It is also possible to use extreme learning machines through the `elm` function.

forecast This package contains the `nnetar` function used to forecast a time series using a multi-layer perceptron with a hidden layer.

Table 2 shows a comparison of these packages in terms of the following features:

- Is it possible to use arbitrary autoregressive lags? In some packages, for example, the lags have to be consecutive values.

- Is it possible to generate the forecasts indicating only the time series and the forecasting horizon?
- Does it include the package tools for plotting the forecasts and other information?
- Is rolling origin evaluation implemented?
- Does the package generate prediction intervals for the forecasts?
- Is it possible to include exogenous variables as predictors in the model?

Table 2: Properties of the packages using machine learning approaches.

	GMDH	**tsDyn**	**nnfor**	**forecast**	**tsfknn**
Arbitrary lags	no	no	yes	no	yes
Default parameters	yes	no	yes	yes	yes
Plotting facilities	no	yes	yes	yes	yes
Rolling origin evaluation	no	no	yes	yes	yes
Prediction intervals	yes	no	no	yes	no
Exogenous variables	no	no	yes	yes	no

We have also conducted a comparison of the methods found in these packages based on forecast accuracy and running time. For this purpose, data from the NN3 forecasting competition (Crone et al., 2011) has been used. In this competition 111 monthly time series of industry data were used. The length of the series ranges from 52 to 120 observations and there is also a balanced mix of seasonal and non-seasonal series. As in the NN3 competition, the next 18 future months of every time series have to be predicted. The MAPE has been used to assess the forecast accuracy. Given the forecast F for a NN3 time series with actual values X:

$$\text{MAPE} = \frac{100}{18} \sum_{t=1}^{18} \left| \frac{X_t - F_t}{X_t} \right|$$

Given a certain method, its MAPE is computed for the 111 time series and averaged in order to obtain a global MAPE. This global MAPE appears in the first row of Table 3 for the different methods. In the comparison the package **GMDH** has not been included because at most it allows users to forecast 5-steps ahead. `elm` and `mlp` are functions from the **nnfor** package for computing extreme learning machines and multi-layer perceptrons respectively. `auto.arima` and `ets` are functions belonging to the **forecast** package that implement ARIMA and exponential smoothing. When calling the functions we have specified as few parameters as possible, so that the function selects automatically or by default the value of the parameters. The statistical models have achieved the best results. Among the machine learning approaches our package is the winner.

In the second row of Table 3 the time in seconds needed for fitting the model and generating the forecasts is shown. There are significant differences between the methods, with our package being one of the fastest methods.

Table 3: Properties of the packages using machine learning approaches.

	tsDyn	`elm`	`mlp`	`nnetar`	`auto.arima`	`ets`	**tsfknn**
MAPE	20.73	18.76	20.95	18.38	15.64	15.52	17.06
Time	2	3332	690	15	421	105	4

Functions and methods in the tsfknn package

In this section a succinct description of all the functions and methods in the **tsfknn** package is given. Most of the functions have already been described above. For those functions not mentioned above a brief example of use is given:

`knn_forecasting` given a time series and some metaparameters this function builds a KNN model for forecasting the time series. It also uses this model to make a prediction of the future values of the time series. Information about the model and the prediction is returned in a `"knnForecast"` object.

`knn_examples` shows the examples associated with a KNN model.

`nearest_neighbors` shows the new instance used in a prediction and its nearest neighbors.

`plot and autoplot` plot a time series and its forecast.

`rolling_origin` assesses the forecast accuracy of a KNN model.

`print and summary` show information about a model and its prediction.

`predict` generates new predictions for a given KNN model.

`n_training_examples` indicates the number of examples that a KNN model would have for a given time series and some metaparameters.

Now, a quick example of how to use the functions not explained previously in the paper is given. The methods `print` and `summary` produce the expected result, i.e., they show some information about the model and its prediction:

```
> pred <- knn_forecasting(mdeaths, h = 3)
> print(pred)

Call:  knn_forecasting(timeS = mdeaths, h = 3)

Multiple-Step Ahead Strategy: MIMO
K (number of nearest neighors): 3 models with 3, 5 and 7 neighbors repectively
Autoregressive lags: 1 2 3 4 5 6 7 8 9 10 11 12
Number of examples: 58
Targets are combined using the mean function.
> summary(pred)

Call:  knn_forecasting(timeS = mdeaths, h = 3)

Multiple-Step Ahead Strategy: MIMO
K (number of nearest neighors): 3 models with 3, 5 and 7 neighbors repectively
Autoregressive lags: 1 2 3 4 5 6 7 8 9 10 11 12
Number of examples: 58
Targets are combined using the mean function.
Forecasting horizon: 3
Forecast:
          Jan      Feb      Mar
1980 1990.562 2106.390 1999.143
```

The method `predict` is used to generate new forecasts using a previously fitted model:

```
> pred <- knn_forecasting(mdeaths, h = 3, k = 2, msas = "recursive")
> new_pred <- predict(pred, h = 12)
> print(new_pred$prediction)
        Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct
1980 2141.0 2052.0 1894.0 1477.0 1570.5 1216.5 1130.0 1045.5  991.5 1049.5
        Nov    Dec
1980 1144.5 1520.5
```

Finally, `n_training_examples` is a handy function for knowing how many training example would have a model:

```
> n_training_examples(mdeaths, h = 3, lags = 1:12, msas = "MIMO")
[1] 58
```

Summary

There is hardly any package in R for applying computational intelligence regression methods to time series forecasting. In this paper we have presented the **tsfknn** package that allows users to forecast a time series using KNN regression. The interface of the package is quite simple, allowing users to specify a KNN model and to predict a time series using the model. Furthermore, several plots can be generated illustrating how the prediction has been computed.

Acknowledgment

Funds: This work was partially supported by the project TIN2015-68854-R (FEDER Founds) of the Spanish Ministry of Economy and Competitiveness.

We would like to thank the anonymous reviewers whose comments helped improve and clarify this manuscript.

Bibliography

R. R. Andrawis, A. F. Atiya, and H. El-Shishiny. Forecast combinations of computational intelligence and linear models for the NN5 time series forecasting competition. *International Journal of Forecasting*, 27(3):672 – 688, 2011. URL https://doi.org/10.1016/j.ijforecast.2010.09.005. [p229]

S. Ben Taieb, G. Bontempi, A. F. Atiya, and A. Sorjamaa. A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition. *Expert Syst. Appl.*, 39(8):7067–7083, 2012. URL https://doi.org/10.1016/j.eswa.2012.01.039. [p232]

S. F. Crone, M. Hibon, and K. Nikolopoulos. Advances in forecasting with neural networks? Empirical evidence from the NN3 competition on time series prediction. *International Journal of Forecasting*, 27 (3):635–660, 2011. URL https://doi.org/10.1016/j.ijforecast.2011.04.001. [p239]

O. Dag and C. Yozgatligil. GMDH: An R package for short term forecasting via GMDH-type neural network algorithms. *The R Journal*, 8(1):379–386, 2016. URL https://doi.org/10.32614/RJ-2016-028. [p238]

M. Hibon and T. Evgeniou. To combine or not to combine: Selecting among forecasts and their combinations. *International Journal of Forecasting*, 21(1):15–24, 2005. URL https://doi.org/10.1016/j.ijforecast.2004.05.002. [p234]

R. Hyndman and Y. Khandakar. Automatic time series forecasting: The forecast package for R. *Journal of Statistical Software*, 27(1):1–22, 2008. URL https://doi.org/10.18637/jss.v027.i03. [p229]

R. J. Hyndman and G. Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, 2014. URL https://www.otexts.org/book/fpp. [p229]

R. J. Hyndman and B. Billah. Unmasking the Theta method. *International Journal of Forecasting*, 19(2): 287 – 290, 2003. URL https://doi.org/10.1016/S0169-2070(01)00143-1. [p229]

R. J. Hyndman and A. B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, pages 679–688, 2006. URL https://doi.org/10.1016/j.ijforecast.2006.03.001. [p234]

N. Kourentzes. *Nnfor: Time Series Forecasting with Neural Networks*, 2017. URL https://CRAN.R-project.org/package=nnfor. R package version 0.9.2. [p229]

M. Kuhn. Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5): 1–26, 2008. URL https://doi.org/10.18637/jss.v028.i05. [p229]

F. Martínez, M. P. Frías, M. D. Pérez, and A. J. Rivera. A methodology for applying k-nearest neighbor to time series forecasting. *Artificial Intelligence Review*, 2017. URL https://doi.org/10.1007/s10462-017-9593-z. [p229, 234, 235]

F. Martínez, M. P. Frías, M. D. Pérez, and A. J. Rivera. Dealing with seasonality by narrowing the training set in time series forecasting with kNN. *Expert Systems with Applications*, 103:38–48, 2018. URL https://doi.org/10.1016/j.eswa.2018.03.005. [p235]

D. Shaub and P. Ellis. *forecastHybrid: Convenient Functions for Ensemble Time Series Forecasts*, 2019. URL https://CRAN.R-project.org/package=forecastHybrid. R package version 4.2.17. [p238]

X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14(1):1–37, 2007. URL https://doi.org/10.1007/s10115-007-0114-2. [p229]

W. Yan. Toward automatic time-series forecasting using neural networks. *IEEE Trans. Neural Netw. Learning Syst.*, 23(7):1028–1039, 2012. URL https://doi.org/10.1109/TNNLS.2012.2198074. [p235]

G. Zhang, B. Eddy Patuwo, and M. Y. Hu. Forecasting with artificial neural networks:: The state of the art. *International Journal of Forecasting*, 14(1):35–62, 1998. URL https://doi.org/10.1016/S0169-2070(97)00044-7. [p229]

Francisco Martínez
Computer Science Department
University of Jaén
Spain
fmartin@ujaen.es

María P. Frías
Statistics and Operations Research Department
University of Jaén
Spain
mpfrias@ujaen.es

Francisco Charte
Computer Science Department
University of Jaén
Spain
fcharte@ujaen.es

Antonio J. Rivera
Computer Science Department
University of Jaén
Spain
arivera@ujaen.es