SIMEXBoost: An R package for Analysis of High-Dimensional Error-Prone Data Based on Boosting Method

by Li-Pang Chen and Bangxu Qiu

Abstract Boosting is a powerful statistical learning method. Its key feature is the ability to derive a strong learner from simple yet weak learners by iteratively updating the learning results. Moreover, boosting algorithms have been employed to do variable selection and estimation for regression models. However, measurement error usually appears in covariates. Ignoring measurement error can lead to biased estimates and wrong inferences. To the best of our knowledge, few packages have been developed to address measurement error and variable selection simultaneously by using boosting algorithms. In this paper, we introduce an R package SIMEXBoost, which covers some widely used regression models and applies the simulation and extrapolation method to deal with measurement error effects. Moreover, the package SIMEXBoost enables us to do variable selection and estimation for high-dimensional data under various regression models. To assess the performance and illustrate the features of the package, we conduct numerical studies.

1 Introduction

In statistical analysis, regression models are important methods for characterizing the relationship between response and the covariates. When the response follows exponential family distributions, generalized linear models (GLM) are commonly used to link the response and the covariates. If the response is taken as failure time and is incomplete due to censoring (e.g., Lawless, 2003), the accelerated failure time model (AFT) might be one of useful strategies to characterize the survival outcome in survival analysis. In recent years, complex modeling structures have been explored when building GLM or survival models, including semi-parametric or mixed-effects structures. To address these challenges, several statistical learning methods, including the boosting approaches (e.g., Hastie et al., 2008), have been developed.

In the contemporary statistical analysis, researchers may frequently encounter high-dimensionality in variables. In particular, high-dimensional data may contain many irrelevant covariates that may affect analysis results. Therefore, it is crucial to do variable selection. In the development of statistical methods, some useful strategies have been proposed, such as regularization approaches (e.g., Tibshirani, 1996; Zou, 2006; Zou and Hastie, 2005) or feature screening methods (e.g., Chen, 2021; Chen, 2023b). In addition, Wolfson (2011) and Brown et al. (2017) proposed the boosting method to do variable selection, which avoids having to deal with non-differentiable penalty functions. The other important feature is measurement error in variables, which is ubiquitous in applications. Moreover, ignoring measurement error effects may affect the estimation results (e.g., Chen and Yi, 2021). Therefore, it is necessary to correct for measurement error effects. A large body of methods has been well established to address variable selection, correction of measurement error, or both. Recently, Chen (2023c) developed the SIMEX method and the regression calibration method with the boosting algorithm accommodated to handle variable selection and measurement error correction for GLMs. Chen and Qiu (2023) considered the AFT model to fit time-to-event responses and proposed the SIMEX method to address measurement error. Chen (2023d) derived the corrected estimating function based on the logistic regression or probit models and applied the boosting method to do variable selection for binary outcomes.

In applications, several commonly used packages associated with existing methods have been developed for public use. A detailed list is summarized in Table 1. Specifically, with variable selection and measurement error ignored, most packages related to boosting methods, including **bst** and **adabag**, can handle classification with binary or multi-class responses. Some packages can deal with different response families. For example, **xgboost** and **lightgbm** can deal with continuous responses; **gbm** and **gamboostLSS** can be used to model survival data under the Cox model and the AFT model, respectively; **GMMBoost** is useful for handling mixed-effects models. In the presence of high-dimensional but precisely measured variables, **glmnet** and **SIS** are two popular packages for variable selection or feature screening, respectively. On the contrary, if variables are subject to measurement error, the two packages **GLSME** and **mecor** focus on linear models and aim to adjust for measurement error effects in the response and/or covariates. Moreover, the simulation and extrapolation (SIMEX) method (e.g., Chen and Yi, 2021; Carroll et al., 2006) has been a powerful strategy to correct for measurement error effects, and has been widely used under several types of regression models in

existing R packages, including **simex** for GLM, **augSIMEX** for GLM with error-prone continuous and discrete variables, and **simexaft** for the AFT model. In addition to the R software, Chen (2023a) developed a Python package **BOOME** to handle variable selection and measurement error for binary outcomes.

Table 1: Comparisons among existing and proposed packages. This table summarizes three categories of packages: (i) variable selection without measurement error correction (**glmnet**, **SIS**), (ii) measurement error correction without variable selection (**GLSME**, **mecor**, **augSIMEX**, **simex**, **simexaft**), (iii) statistical learning approaches that handle estimation without consideration of measurement error and variable selection (**bst**, **xgboost**, **gbm**, **adabag**, **lightgbm**, **GMMBoost**, **gamboostLSS**). The proposed package **SIMEXBoost** is included in these three categories. In the Usage heading, 'LM' denotes the linear model; 'Class' is the classification; 'Pois' is the Poisson regression, 'Cox' is the Cox model; 'AFT' is the AFT model; 'SL' is statistical learning; 'ME' represents measurement error correction; 'VS' is variable selection; and 'Col' represents collinearity.

	Usage								
Packages	LM	Class[1]	Pois	Cox	AFT	SL[2]	ME	VS	Col
SIMEXBoost Qiu and Chen (2023)	✓	✓	✓	×	✓	✓	✓	✓	✓
glmnet Friedman et al. (2023)	✓	✓	✓	✓	×	×	×	✓	✓
SIS Feng et al. (2020)	✓	✓	✓	✓	×	×	×	✓	×
GLSME Bartoszek (2019)	✓	×	×	×	×	×	✓	×	×
mecor Nab (2021)	✓	×	×	×	×	×	✓	×	×
augSIMEX Zhang and Yi (2020)	✓	✓	✓	×	×	×	✓	×	×
simex Lederer et al. (2019)	✓	✓	✓	✓	×	×	✓	×	×
simexaft Xiong et al. (2019)	×	×	×	×	✓	×	✓	×	×
bst Wang and Hothorn (2023)	×	✓	×	×	×	✓	×	×	×
xgboost Chen et al. (2023)	✓	✓	×	×	×	✓	×	×	×
gbm Greenwell et al. (2022)	✓	✓	✓	✓	×	✓	×	×	×
adabag Alfaro et al. (2023)	×	✓	×	×	×	✓	×	×	×
lightgbm Shi et al. (2023)	✓	✓	×	×	×	✓	×	×	×
GMMBoost Groll (2020)	✓	✓	×	×	×	✓	×	×	×
gamboostLSS Hofner et al. (2023)	✓	×	✓	×	✓	✓	×	×	×
BOOME (in Python) Chen (2023a)	×	✓	×	×	×	✓	✓	✓	✓

[1] "Class" includes binary or multiclass classification, and the construction of logistic regression models.
[2] "SL" contains several estimation methods based on machine learning approaches, such as tree and random forest. In addition, the corresponding packages may handle complex structures, including semi-paramatric models, mixed-effects models, and generalized additive models.

While many packages have been available to handle either variable selection or measurement error correction, few packages deal with these two features simultaneously. To address those concerns, we develop the R package **SIMEXBoost** (Qiu and Chen, 2023) by extending the method in Chen (2023c) and Chen and Qiu (2023), which covers commonly used GLM and AFT models. Motivated by the idea of the boosting algorithm (see e.g., Hastie et al., 2008, Section 16.2), **SIMEXBoost** aims to use estimating functions to iteratively retain informative covariates and exclude unimportant ones, yielding variable selection result. In addition, to deal with measurement error effects, the package **SIMEXBoost** primarily employs the SIMEX method to efficiently correct for measurement error effects for different types of regression models. There are several advantages of **SIMEXBoost** over the existing packages. Specifically, as summarized in Table 1, while **glmnet** and **SIS** are able to handle variable selection, they fail to deal with measurement error effects. In addition, the two packages **GLSME** and **mecor** focus on linear models and aim to adjust for measurement error effects in the response and/or covariates, but they cannot deal with variable selection. On the contrary, the contribution of the package **SIMEXBoost** is able handle measurement error in variables, and do variable selection and estimation simultaneously. Moreover, boosting iteration may reduce the possibility of falsely excluding important covariates and enhance the accuracy of the estimator. Most importantly, **SIMEXBoost** is

able to deal with collinearity of variables by using the L_2-norm penalty function.

The remainder is organized as follows. In the second section, we introduce the data structure and the corresponding regression models. In addition, the boosting algorithm is outlined. In the third section, we introduce the measurement error model and extend the correction of measurement error effects to the boosting algorithm. In the fourth section, we introduce functions and their arguments in the R package **SIMEXBoost**. In the fifth section, we demonstrate the application of the R package **SIMEXBoost** and conduct simulation studies to assess the performance of the boosting estimators. Moreover, we also implement **SIMEXBoost** in a real dataset. A general discussion is presented in the last section. The supporting information, including a real dataset, programming code, and numerical results in csv files, are placed in the corresponding author's GitHub, whose link is given by `https://github.com/lchen723/SIMEXBoost.git`.

2 Notation, Models, and Boosting Procedure

2.1 Model

Let Y denote the response, and let $\mathbf{X}$ be the p-dimensional vector of covariates. Suppose that we have a sample of n subjects and for $i = 1, \cdots, n$, $\{Y_i, \mathbf{X}_i\}$ has the same distribution as $\{Y, \mathbf{X}\}$.

Let $\boldsymbol{\beta}$ be a p-dimensional vector of (unknown) parameters associated with the covariates $\mathbf{X}$, and write $\mathbf{X}^\top \boldsymbol{\beta}$ as the linear predictor. In the framework of statistical learning, to characterize the relationship between Y and $\mathbf{X}$, a commonly used approach is to link Y and $\mathbf{X}^\top \boldsymbol{\beta}$ through the convex loss function $L : \mathbb{S} \times \mathbb{R} \to \mathbb{R}$, where $\mathbb{S}$ is the support of Y. Let the risk function be defined as the expectation of the loss function, i.e., $R(\boldsymbol{\beta}) \triangleq E\left\{L(Y, \mathbf{X}^\top \boldsymbol{\beta})\right\}$. Under the finite sample size n, the empirical version of $R(\boldsymbol{\beta})$ is given by

$$\frac{1}{n}\sum_{i=1}^{n} L(Y_i, \mathbf{X}_i^\top \boldsymbol{\beta}).$$

Our goal is to estimate $\boldsymbol{\beta}$ by minimizing the risk function, and the resulting estimator is given by

$$\widehat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left\{ \frac{1}{n}\sum_{i=1}^{n} L(Y_i, \mathbf{X}_i^\top \boldsymbol{\beta}) \right\}.$$

Equivalently, $\widehat{\boldsymbol{\beta}}$ satisfies the estimating equation $\mathbf{g}(\mathbf{X}, \boldsymbol{\beta}) = 0$, where $\mathbf{g}(\mathbf{X}, \boldsymbol{\beta})$ is the estimating function of $\boldsymbol{\beta}$, defined as the first order derivative of $\frac{1}{n}\sum_{i=1}^{n} L(Y_i, \mathbf{X}_i^\top \boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$.

2.2 Boosting Procedure

High-dimensionality and sparsity of $\boldsymbol{\beta}$ are crucial concerns, which reflect the idea that some covariates are not informative with respect to Y. To address these issues and provide a reliable estimator of $\boldsymbol{\beta}$, we employ the boosting procedure to perform variable selection and estimation (e.g., Hastie et al., 2008). This version of the boosting algorithm is motivated by Wolfson (2011) and Brown et al. (2017), and is applied to handle GLM (Chen, 2023c) as well as AFT models (Chen and Qiu, 2023). An overall procedure is presented in Algorithm 1 with three key steps. Specifically, Steps 1 and 2 in Algorithm 1 treat the estimating function evaluated at an iterated value as the signal, and use it to determine informative indexes of covariates and parameters. Noting that there is a parameter τ in Step 2 that is used to control the number of selected covariates in each iteration. In our numerical studies, $\tau = 0.9$ seems to be a suitable choice and has a satisfactory performance.

After that, Step 3 in Algorithm 1 updates $\boldsymbol{\beta}$ using the informative indexes determined in Step 2 by the sign of signals with increment κ. This approach follows the steepest descent method (see e.g., Boyd and Vandenberghe, 2004, Section 9.4.2) and can be used to deal with the L_1-norm for variable selection. In addition, as discussed in Section 16.2.1 of Hastie et al. (2008), the value κ has the opposite relationship with the number of iteration M that smaller κ requires larger M. In our consideration, we specify $\kappa = 0.05$. Finally, repeating the iteration M times gives the desired estimator. With M time iterations, we can obtain the final set $\mathcal{J}_M$ containing informative covariates and the corresponding $\boldsymbol{\beta}^{(M)} = (\beta_1^{(M)}, \cdots, \beta_p^{(M)})^\top$, accomplishing variable selection.

Algorithm 1: Boost_VSE Algorithm

Let $\boldsymbol{\beta}^{(0)} = 0$ denote an initial value;
for *iteration m with* $m = 0, 1, 2, \cdots, M$ **do**
Step 1: calculate $\boldsymbol{\Delta}^{(m-1)} = \mathbf{g}(\mathbf{X}, \boldsymbol{\beta})\big|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(m-1)}}$;
Step 2: determine $\mathcal{J}_m = \left\{ j : \left|\Delta_j^{(m-1)}\right| \geq \tau \max_{j'} \left|\Delta_{j'}^{(m-1)}\right| \right\}$;
Step 3: update $\beta_j^{(m)} \leftarrow \beta_j^{(m-1)} + \kappa \cdot \text{sign}(\Delta_j^{(m-1)})$ for all $j \in \mathcal{J}_m$, and define $\boldsymbol{\beta}^{(m)} = (\beta_1^{(m)}, \cdots, \beta_p^{(m)})^\top$;

Finally, for the application of Algorithm 1, we consider some specific models and the corresponding regression models listed below. With models specified, we can further determine the estimating function $\mathbf{g}(\mathbf{X}, \boldsymbol{\beta})$.

Linear regression models:
Given the dataset $\{\{Y_i, \mathbf{X}_i\} : i = 1, \cdots, n\}$ with Y_i being a continuous and univariate random variable, linear models are characterized as

$$Y_i = \mathbf{X}_i^\top \boldsymbol{\beta} + \epsilon_i \tag{1}$$

where ϵ_i is the noise term with $E(\epsilon_i) = 0$ and $\text{var}(\epsilon_i) = \sigma_\epsilon^2$. The estimating function is defined as the first order derivative of the least squares function:

$$\mathbf{g}(\mathbf{X}, \boldsymbol{\beta}) = \sum_{i=1}^{n} -\mathbf{X}_i (Y_i - \mathbf{X}_i^\top \boldsymbol{\beta}). \tag{2}$$

Logistic regression models:
If Y_i is a binary and univariate random variable, then Y_i and $\mathbf{X}_i$ are usually characterized by a logistic regression model:

$$\pi_i = \frac{\exp\left(\mathbf{X}_i^\top \boldsymbol{\beta}\right)}{1 + \exp\left(\mathbf{X}_i^\top \boldsymbol{\beta}\right)}, \tag{3}$$

where $\pi_i \triangleq P(Y_i = 1 | \mathbf{X}_i)$. Following the idea in Agresti (2012), we can construct the likelihood function based on (3). Therefore, the resulting estimating function is given by the first order derivative of the likelihood function:

$$\mathbf{g}(\mathbf{X}, \boldsymbol{\beta}) = -\sum_{i=1}^{n} \mathbf{X}_i \left\{ Y_i - \frac{\exp\left(\mathbf{X}_i^\top \boldsymbol{\beta}\right)}{1 + \exp\left(\mathbf{X}_i^\top \boldsymbol{\beta}\right)} \right\}. \tag{4}$$

Poisson regression models:
When Y_i is a count and univariate random variable, one can adopt the Poisson regression model to fit Y_i and $\mathbf{X}_i$:

$$\log \lambda_i = \mathbf{X}_i^\top \boldsymbol{\beta} \tag{5}$$

where λ_i is the parameter of the Poisson distribution. Following the framework of generalized linear models, the likelihood function based on (5) can be determined. Therefore, the first order derivative of the likelihood function under (5) yields the corresponding estimating function, which is given by

$$\mathbf{g}(\mathbf{X}, \boldsymbol{\beta}) = -\sum_{i=1}^{n} \mathbf{X}_i \left\{ Y_i - \exp\left(\mathbf{X}_i^\top \boldsymbol{\beta}\right) \right\}. \tag{6}$$

Accelerated failure time models:
In survival analysis, the response is known as the failure time, denoted $\widetilde{T}_i > 0$, and the accelerated failure time (AFT) model is a commonly used model for characterizing the relationship between the survival time and the covariates (e.g., Lawless, 2003). Specifically, the AFT model is formulated as

$$\log \widetilde{T}_i = \mathbf{X}_i^\top \boldsymbol{\beta} + \eta_i, \tag{7}$$

where η_i is the noise term of (7). In the framework of survival analysis, the main challenge is that $\widetilde{T}_i$ is usually incomplete due to how the observations are collected. In particular, in this study, the failure time may suffer from *length-biased* and *interval-censoring*, which cause the data

to be biased and incomplete.

Specifically, for the length-biased sampling, it is common to assume that the incidence rate of the initial event is constant over calendar time, and the truncation time, denoted $\widetilde{A}_i$, is uniformly distributed in $[0, \xi]$, where ξ is the maximum support of $\widetilde{T}_i$ (e.g., Chen and Qiu, 2023). For the length-biased data, we can observe $(\widetilde{A}_i, \widetilde{T}_i)$ only if $\widetilde{T}_i \geq \widetilde{A}_i$, and thus, we denote $(T_i, A_i) \equiv (\widetilde{T}_i, \widetilde{A}_i) | \widetilde{T}_i \geq \widetilde{A}_i$ as the observed version of $(\widetilde{T}_i, \widetilde{A}_i)$.

On the other hand, for the observed T_i, we may encounter the interval-censoring. Suppose that T_i is not exactly observed but only determined at a sequence of examination times, denoted as $A_i = U_0 < U_1 < \cdots < U_N \leq \xi$ for some constant $N > 0$. The failure time is then known to lie in the interval (L, R), where $L_i = \max\{U_k : U_k < T_i, k = 0, \cdots, N\}$ and $R_i = \min\{U_k : U_k \geq T_i, k = 1, \cdots, N+1\}$ with $U_{N+1} \triangleq \infty$. Moreover, if T_i occurs before the first examination time, then $(L_i, R_i) \triangleq (A_i, U_1)$; if the failure has not occurred at the last examination time, then $(L_i, R_i) \triangleq (U_N, \infty)$. Finally, let Δ_i denote the indicator, where a value 1 indicates that T_i is observed and zero otherwise. As a consequence, for a sample with size n, the length-biased and interval-censored survival data is given by $\{\{A_i, \Delta_i, Y_i, X_i\} : i = 1, \cdots, n\}$ with $Y_i \triangleq \{\Delta_i T_i, (1-\Delta_i) L_i, (1-\Delta_i) R_i\}$.

Based on the length-biased and interval-censored data, we can construct the estimating function (e.g., Chen and Qiu, 2023)

$$\mathbf{g}(\mathbf{X}, \boldsymbol{\beta}) = \sum_{i=1}^{n} \mathbf{X}_i \left\{ \Delta_i \frac{Y_{\beta,i}}{\exp(Y_{\beta,i})} + (1-\Delta_i) \frac{\int_{L_{i,0}}^{R_{i,0}} u^{-1} \log u dF_0(u)}{F_0(R_{i,0}) - F_0(L_{i,0})} \right\}, \tag{8}$$

where $Y_{\beta,i} = \log T_i - \mathbf{X}_i^\top \boldsymbol{\beta}$, $R_{i,0} = R_i \exp(-\mathbf{X}_i^\top \beta)$, $L_{i,0} = L_i \exp(-\mathbf{X}_i^\top \beta)$, and F_0 is the cumulative distribution function of η_i.

3 A Modified Boosting Method with the Presence of Covariate Measurement Error

3.1 Measurement Error Models

For $i = 1, \cdots, n$, let $\mathbf{X}_i^*$ denote the surrogate, or observed covariate, of $\mathbf{X}_i$. Let $\boldsymbol{\Sigma}_{X^*}$ and $\boldsymbol{\Sigma}_X$ be the $p \times p$ covariance matrices of $\mathbf{X}_i^*$ and $\mathbf{X}_i$, respectively. In our development, we focus on the classical measurement error model (e.g., Carroll et al., 2006, Chapter 1):

$$\mathbf{X}_i^* = \mathbf{X}_i + \mathbf{e}_i, \tag{9}$$

where $\mathbf{e}_i$ is independent of $\{\mathbf{X}_i, Y_i\}$ and ϵ_i in (1), $\mathbf{e}_i$ follows a normal distribution with mean zero and the covariance matrix $\boldsymbol{\Sigma}_e$, say $N(0, \boldsymbol{\Sigma}_e)$. Noting that the covariance matrix $\boldsymbol{\Sigma}_e$ is usually unknown. To determine it, we can either employ sensitivity analyses to reasonably specify values, or directly estimate it if additional information, such as repeated measurements or validation sample is available (see e.g., Chen and Yi, 2021). As a result, in the presence of measurement error, the *observed* dataset is now given by $\{\{Y_i, \mathbf{X}_i^*\} : i = 1, \cdots, n\}$.

3.2 Boosting with Measurement Error Correction

In the presence of measurement error, it is known that directly using $\mathbf{X}_i^*$ in the estimating procedure without the correction of measurement error effects may incur biased estimate and wrong conclusion (see e.g., Carroll et al., 2006). Therefore, even though Algorithm 1 is valid to estimate $\boldsymbol{\beta}$ for regression models in the 'Boosting Procedure' subsection, it is insufficient in the presence of measurement error effects.

To deal with measurement error in covariates, we extend Algorithm 1 by adopting the SIMEX method to eliminate the impact of measurement error (e.g., Chen and Yi, 2021). The modified algorithm, called *SIMEXBoost*, is summarized in Algorithm 2.

The idea of the SIMEX method is to first establish the trend of measurement error-induced biases as a function of the variance of measurement error by artificially creating a sequence of surrogate measurements, and then extrapolate this trend back to the case without measurement error. Specifically, in Step 1 of Algorithm 2, we artificially create a sequence of error-contaminated surrogate measurements by introducing different degrees of measurement error. After that, as shown in Step 2 of Algorithm 2, we apply those surrogate measurements to the boosting procedure, and use a new function $\mathbf{g}_{\text{SIM}}(\boldsymbol{\beta}; b, \zeta)$, which is defined as (2), (4), (6), or (8) with $\mathbf{X}_i$ replaced by $\mathbf{W}_i(b, \zeta)$ defined in

(10), to obtain biased estimates by running an estimation method developed for error-free settings. Finally, Step 3 in Algorithm 2 traces the pattern of biased estimates against varying magnitudes of measurement error and then does extrapolation based on linear or quadratic regression models.

Noting that there are several parameters in Algorithm 2. The parameters M, κ, and τ in Step 2 are the same as those in Algorithm 1. On the other hand, Step 1 contains a value B and a sequence of $\mathcal{Z}$ that are usually user-specified. Typically, $\mathcal{Z}$ is usually defined as K equal-width cutpoints in an interval $[0, 1]$ for some positive constant K. A value B is used to perform the Monte Carlo computation in (11) and make the estimator more stable. A larger value of B may implicitly incur longer computational time. Our numerical experiments show that $B = 50$ gives satisfactory performance.

Algorithm 2: SIMEXBoost Algorithm

Step 1: Generate the working data $\mathbf{W}_i(b, \zeta)$ by

$$\mathbf{W}_i(b, \zeta) = \mathbf{X}_i^* + \sqrt{\zeta}\mathbf{e}_{i,b} \tag{10}$$

for $b = 1, \cdots, B$ and $\zeta \in \mathcal{Z}$, where $\mathbf{e}_{i,b} \sim N(0, \mathbf{\Sigma}_e)$ independently.

Step 2: Boosting estimation.
Perform the following boosting procedure with M iterations.

for $b = 1, \cdots, B$ *and* $\zeta \in \mathcal{Z}$ **do**
  Let $\boldsymbol{\beta}^{(0)}(b, \zeta) = \mathbf{0}$ denote an initial value;
  for *iteration* m *with* $m = 0, 1, 2, \cdots, M$ **do**
    Step 2.1: calculate $\boldsymbol{\Delta}^{(m-1)}(b, \zeta) = \mathbf{g}_{\text{SIM}}(\boldsymbol{\beta}; b, \zeta)\big|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(m-1)}}$;
    Step 2.2: determine $\mathcal{J}_m(b, \zeta) = \left\{ j : \left|\Delta_j^{(m-1)}(b, \zeta)\right| \geq \tau \max_j \left|\Delta_j^{(m-1)}(b, \zeta)\right| \right\}$;
    Step 2.3: update $\beta_j^{(m)}(b, \zeta) \leftarrow \beta_j^{(m-1)}(b, \zeta) + \kappa \cdot \text{sign}(\Delta_j^{(m-1)}(b, \zeta))$ for all $j \in \mathcal{J}_m(b, \zeta)$;
  Write $\boldsymbol{\beta}^{(M)}(b, \zeta) = \left(\beta_1^{(M)}(b, \zeta), \cdots, \beta_p^{(M)}(b, \zeta)\right)^\top$;

When $\boldsymbol{\beta}^{(M)}(b, \zeta)$ is obtained for $b = 1, \cdots, B$ and $\zeta \in \mathcal{Z}$, compute an average

$$\boldsymbol{\beta}^{(M)}(\zeta) = \frac{1}{B}\sum_{b=1}^{B} \boldsymbol{\beta}^{(M)}(b, \zeta) \text{ for } \zeta \in \mathcal{Z}. \tag{11}$$

Step 3: Extrapolation.
Fit a sequence $\left\{ \left(\zeta, \boldsymbol{\beta}^{(M)}(\zeta)\right) : \zeta \in \mathcal{Z} \right\}$ by a regression model, and the final value is given by the extrapolated value at $\zeta = -1$.

4 Description and Implementation of SIMEXBoost

We develop an R package, called **SIMEXBoost**, to implement the variable selection and estimation with measurement error correction described in the preceding section. This package depends on the **MASS** package only. The package **SIMEXBoost** contains three functions: `ME_Data`, `Boost_VSE`, and `SIMEXBoost`. The function `ME_Data` aims to generate artificial data under specific models listed in 'Boosting Procedure' subsection and error-prone covariates. The function `Boost_VSE` implements the boosting procedure in Algorithm 1, and the function `SIMEXBoost` implements the error-eliminated boosting procedure as displayed in Algorithm 2. We now describe the details of these three functions.

`ME_Data`

We use the following command to obtain the artificial data:

```
ME_Data(X,beta,type="normal",sigmae,pr0=0.5)
```

where the meaning of each argument is described as follows:

- `X`: An $n \times p$ matrix with components generated by random variables. It is provided by the user.
- `beta`: A p-dimensional vector of parameters, which is specified by the user.
- `type`: A regression model that is specified to generate the response. Some choices listed in 'Boosting Procedure' subsection are provided in this argument. `normal` means the linear

regression model (1) with the error term generated by the standard normal distribution; `binary` means the logistic regression model (3); `poisson` means the Poisson regression model (5). In addition, the accelerated failure time (AFT) model is considered to fit length-biased and interval-censored survival data. Specifically, `AFT-normal` generates the length-biased and interval-censored survival data under the AFT model (7) with the error term being normal distributions; `AFT-loggamma` generates the length-biased and interval-censored survival data under the AFT model with the error term being log-gamma distributions.

- `sigmae`: A $p \times p$ covariance matrix $\mathbf{\Sigma}_e$ in the measurement error model (9). Given $\mathbf{\Sigma}_e$ with non-zero entries, by (9), one can generate the error-prone covariates $\mathbf{X}_i^*$. Moreover, if $\mathbf{\Sigma}_e$ is given by the zero matrix, then $\mathbf{e}_i$ is generated as zero values, yielding that $\mathbf{X}_i^*$ is equal to $\mathbf{X}_i$, and thus, the resulting covariate is the original input given by users.
- `pr0`: A numerical value in an interval $(0, 1)$. It is used to determine the censoring rate for the length-biased and interval-censored data.

The function `ME_Data` returns a list of components:

- `response`: It gives the response generated by a specific regression model. `type="normal"` gives a n-dimensional continuous vector; `type="binary"` gives a n-dimensional vector with binary entries; `type="poisson"` gives a n-dimensional vector with entries being counting numbers. In addition, `type="AFT-normal"` and `type="AFT-loggamma"` generates a $n \times 2$ matrix of length-biased and interval-censored responses, where the first column is the lower bound of an interval-censored response and the second column is the upper bound of an interval-censored response.
- `ME_covariate`: This output gives a $n \times p$ matrix of "error-prone" or "precisely measured" covariates. Specifically, as discussed in an argument `sigmae`, if $\mathbf{\Sigma}_e$ is a non-zero covariance matrix, then the result of `ME_covariate` is given by $\mathbf{X}_i^*$; if $\mathbf{\Sigma}_e$ is a zero matrix, then the result of `ME_covariate` is the original input, say $\mathbf{X}_i$.

`Boost_VSE`

We use the following function to perform Algorithm 1:

```
Boost_VSE(Y,Xstar,type="normal",Iter=200,Lambda=0)
```

where the meaning of each argument is described as follows:

- `Y`: The response variable. If `type` is specified as `normal`, `binary`, or `poisson`, then `Y` should be a n-dimensional vector; if `type` is given by `AFT-normal` or `AFT-loggamma`, then `Y` should be a $n \times 2$ matrix of interval-censored responses, where the first column is the lower bound of an interval-censored response and the second column is the upper bound of an interval-censored response.
- `Xstar`: This argument needs a $n \times p$ matrix of covariates. It can be error-prone or precisely measured.
- `type`: This argument specifies the regression models as previously described in the function `ME_Data` as well as their corresponding estimating functions given by (2), (4), (6), and (8).
- `Iter`: The number of iterations M for the boosting procedure in Algorithm 1.
- `Lambda`: A tuning parameter that aims to deal with the collinearity of covariates. `Lambda=0` means that no L_2-norm is involved, and it is the default value.

The function `Boost_VSE` returns a list of components:

- `BetaHat`: The vector of estimated coefficient obtained by Algorithm 1. In particular, if the covariate `Xstar` is generated by `ME_Data` with the argument `sigmae` being the zero matrix, then the resulting vector is the "ordinary" estimator based on the boosting procedure; if the covariate `Xstar` is generated by `ME_Data` with the argument `sigmae` being a non-zero covariance matrix, then we call the resulting vector as the *naive* estimator due to involvement of measurement error effects without correction.

`SIMEXBoost`

Basically, some arguments in this function are the same as `Boost_VSE`, except for some slightly different requirements and additional arguments that are related to the SIMEX method.

We use the following function to perform Algorithm 2:

```
SIMEXBoost(Y,Xstar,zeta=c(0,0.25,0.5,0.75,1),B=500,type="normal",sigmae,
           Iter=100, Lambda=0, Extrapolation="linear")
```

where the meanings of arguments `Y`, `Xstar`, `type`, `Iter`, and `Lambda` are the same as those in the function `Boost_VSE`; additional arguments `zeta`, `B`, `sigmae`, and `Extrapolation`, which are used to implement the SIMEX method to correct for measurement error effects, are described as follows:

- `zeta`: The user-specific sequence of values described as $\mathcal{Z}$ in Step 1 of Algorithm 2.
- `B`: The user-specific positive number of repetition described as B in Step 1 of Algorithm 2.
- `sigmae`: An $p \times p$ covariance matrix $\mathbf{\Sigma}_e$ in the measurement error model (9). In practical applications, if auxiliary information is unavailable, sensitivity analyses can be adopted to reasonably specify values of $\mathbf{\Sigma}_e$. If additional information, such as repeated measurements or validation samples, is available, one can directly estimate $\mathbf{\Sigma}_e$.
- `Extrapolation`: A extrapolation function for the SIMEX method implemented to Step 3 of Algorithm 2. In the framework of the SIMEX method, quadratic and linear functions are common. Therefore, in this argument, we provide two choices of the extrapolation functions, `linear` and `quadratic`.

The function `SIMEXBoost` returns a list of components:

- `BetaHatCorrect`: The resulting vector of corrected estimates obtained by Algorithm 2.

5 Numerical Studies and Demonstration of Programming Code

In this section, we demonstrate the usage of the functions in the package **SIMEXBoost**. There are two parts in this section: we first illustrate simulation studies for linear regression, Poisson regression, and AFT models. After that, we apply the package to analyze a real-world dataset with binary responses based on a logistic regression model.

5.1 Simulation Studies

In this section, we use simulation studies to demonstrate applications of functions in the package **SIMEXBoost** and assess the performance of the estimators derived by two functions `Boost_VSE` and `SIMEXBoost`.

We consider the dimension of covariates $p = 200$ or 500, and let the sample size $n = 400$. Let the true value $\boldsymbol{\beta}_0 = (1, 1, 1, \mathbf{0}_{p-3}^\top)^\top$, where $\mathbf{0}_q$ is a q-dimensional zero vector. The unobserved covariate $\mathbf{X}_i$ is generated by the standard normal distribution, and it can be used to generate the response Y_i based on (1), (5), and (7). For the error-prone covariate $\mathbf{X}_i^*$, it can be generated by (9), where $\mathbf{\Sigma}_e$ is a diagonal matrix with common entries σ_e being 0.1, 0.3, and 0.5.

Based on the artificial data $\left\{\{Y_i, \mathbf{X}_i^*\} : i = 1, \cdots, n\right\}$, we first use the function `Boost_VSE` to obtain the estimator *without* measurement error correction, which is called the *naive* estimator. Next, we apply the function `SIMEXBoost` to derive the *corrected* estimator. To assess the performance of variable selection, we examine the specificity (Spe) and the sensitivity (Sen), where the specificity is defined as the proportion of zero coefficients that are correctly estimated to be zero, and the sensitivity is defined as the proportion of non-zero coefficients that are correctly estimated to be non-zero. In addition, to evaluate the performance of estimation, we use the L_1 and L_2-norms to measure bias, which are respectively defined as

$$\left\|\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}_0\right\|_1 = \sum_{j=1}^{p} \left|\widehat{\beta}_j - \beta_{0,j}\right| \text{ and } \left\|\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}_0\right\|_2 = \left\{\sum_{j=1}^{p} \left(\widehat{\beta}_j - \beta_{0,j}\right)^2\right\}^{1/2}, \tag{12}$$

where $\widehat{\boldsymbol{\beta}}$ is the estimator, $\widehat{\beta}_j$ and $\beta_{0,j}$ are the jth component in $\widehat{\boldsymbol{\beta}}$ and $\boldsymbol{\beta}_0$, respectively.

We use two '`for`' loops cover all combinations of p and σ_e (`sigmae`). For each `p` and `sigmae`, we first adapt the function `ME_Data` to generate the simulated data, where `Y` and `Xstar` represent the response and error-prone covariates, respectively. After that, to examine the naive estimator derived by Algorithm 1, we use the function `Boost_VSE` to derive the naive estimator and denote it by `naive`. To implement Algorithm 2, we adopt the function `SIMEXBoost`, where two components $\mathcal{Z}$ and B in Step 1 of Algorithm 2 are specified as `zeta=c(0,0.25,0.5,0.75,1)` and `B=50`, respectively. For the SIMEXBoost method, we also examine linear and quadratic extrapolation functions with the argument `Extrapolation="linear"` or `Extrapolation="quadratic"`, and denote two resulting estimators as

correctL and correctQ, respectively. For the two functions Boost_VSE and SIMEXBoost, we set the number of iterations Iter=50. To save the space in the limited text, we simply illustrate the model (1) with the argument type="normal"; numerical results under other models can be reprocued by the following code with type="normal" replaced by type="poisson" or type="AFT-normal".

Next, we assess the performance of naive, correctL, and correctQ. Given a true vector of parameter beta0, we compute L_1 and L_2-norms in (12) to examine the bias, and compute Spe and Sen to examine variable selection. Under a given p and sigma, biases and variable selection results are recorded by EST1, EST2, and EST3 for the estimators naive, correctL, and correctQ, respectively. Finally, numerical results of three estimators naive, correctL, and correctQ under all settings are summarized by NAIVE, SIMEXL, and SIMEXQ, respectively.

```
library(SIMEXBoost)
library(MASS)

NAIVE = NULL   # naive method
SIMEXL = NULL  # simex method with linear extrapolation function
SIMEXQ = NULL  # simex method with quadratic extrapolation function

for (p in c(200, 500)) {
  for (sigma in c(0.1, 0.3, 0.5)) {
    set.seed(202270)
    beta0 = c(1, 1, 1, rep(0, p - 3))

    X = matrix(rnorm((p) * 400),
               nrow = 400,
               ncol = p,
               byrow = TRUE)

    Sig = diag(sigma ^ 3, dim(X)[2])

    data = ME_Data(
      X = X,
      beta = beta0,
      type = "normal",
      sigmae = Sig
    )
    Y = data$response
    Xstar = data$ME_covariate

    naive = Boost_VSE(Y, Xstar, type = "normal", Iter = 50)$BetaHat

    correctL = SIMEXBoost(
      Y,
      Xstar,
      zeta = c(0, 0.25, 0.5, 0.75, 1),
      B = 50,
      type = "normal",
      sigmae = Sig,
      Iter = 50,
      Lambda = 0,
      Extrapolation = "linear"
    )$BetaHatCorrect
    correctL[which(abs(correctL) < 0.5)] = 0
    correctQ = SIMEXBoost(
      Y,
      Xstar,
      zeta = c(0, 0.25, 0.5, 0.75, 1),
      B = 50,
      type = "normal",
      sigmae = Sig,
      Iter = 50,
      Lambda = 0,
      Extrapolation = "quadratic"
    )$BetaHatCorrect
```

```
    correctQ[which(abs(correctQ) < 0.5)] = 0

    #############

    Sen = which(beta0 != 0)
    Spe0 =  which(beta0 == 0)

    ## results for the naive estimator
    naive = as.numeric(naive)
    L1_norm = sum(abs(naive - beta0))
    L2_norm = sqrt(sum((naive - beta0) ^ 2))
    Spe = length(which(naive[Spe0] == 0)) / length(Spe0)
    Sen = length(which(naive[Sen0] != 0)) / length(Sen0)

    ## results for the error-corrected estimator based on Extrapolation="linear"
    L1_norm_l = sum(abs(correctL - beta0))
    L2_norm_l = sqrt(sum((correctL - beta0) ^ 2))
    Spe_l = length(which(correctL[Spe0] == 0)) / length(Spe0)
    Sen_l = length(which(correctL[Sen0] != 0)) / length(Sen0)

    ## results for the error-corrected estimator based on Extrapolation="quadratic"
    L1_norm_q = sum(abs(correctQ - beta0))
    L2_norm_q = sqrt(sum((correctQ - beta0) ^ 2))
    Spe_q = length(which(correctQ[Spe0] == 0)) / length(Spe0)
    Sen_q = length(which(correctQ[Sen0] != 0)) / length(Sen0)


    #############

    NAIVE = rbind(NAIVE, c(L1_norm, L2_norm, Spe, Sen))
    SIMEXL = rbind(SIMEXL, c(L1_norm_l, L2_norm_l, Spe_l, Sen_l))
    SIMEXQ = rbind(SIMEXQ, c(L1_norm_q, L2_norm_q, Spe_q, Sen_q))
  }
}
```

Numerical results under (1), (5), and (7) are placed in Tables 2-4, respectively. We observe that the naive and corrected estimates are affected by the magnitudes of measurement error effects and the dimension p. When values of `p` and `sigma` become large, the biases given by L_1 and L_2-norms are increasing. For the comparison between the naive and corrected estimators, we can see that biases produced by the naive estimator are significantly larger than those obtained by the corrected estimator. In addition, for the variable selection result, the corrected estimator is able to correctly retain the informative covariates and exclude unimportant ones, except for some cases that one or two covariates may be falsely included. On the contrary, we can observe from the naive method that values of `Spe_naive` are always small while values of `Sen_naive` are equal to one. It indicates that the naive estimator retains the truly important covariates, and meanwhile, includes a lot of unimportant ones, which shows an evidence that the naive method fails to do variable selection. Finally, for the comparison between two extrapolation functions, `Extrapolation="linear"` and `Extrapolation="quadratic"`, we observe that the specification of a linear extrapolation function has slightly better performance than the case under a quadratic extrapolation function, especially when `sigma` is large.

While the proposed SIMEXBoost method can handle measurement error well, the main concern is the computational time. According to the record of the system CPU time (in seconds) from the R function `proc.time()` under the device ASUS DESKTOP-HJSD47S with the processor Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz, we find that, for each setting with fixed `p` and `sigmae`, the proposed SIMEXBoost method requires 14.09 and 15.04 seconds under `Extrapolation="linear"` and `Extrapolation="quadratic"` to derive the estimator for $p = 200$, respectively. Unsurprising, when the dimension increases to $p = 500$, the computational times under `Extrapolation="linear"` and `Extrapolation="quadratic"` are increasing to 17.70 and 19.68 seconds, respectively. On the other hand, without measurement error correction under each setting, the naive method needs 12.98 and 15.71 seconds to derive the estimator for $p = 200$ and 500, yielding the slightly faster computational time than the SIMEXBoost method. It might be due to the measurement error correction with the involvement of $\mathcal{Z}$ and the number of repetitions B in Step 2 of Algorithm 2. As a result, we comment that the SIMEXBoost method is able to address measurement error correction, but a slightly longer computational time is the price that the users should pay for.

Table 2: Simulation results for the model (1) with `type="normal"`. "Naive" is the naive method obtained by the function `Boost_VSE`. "SIMEXBoost-Linear" and "SIMEXBoost-Quadratic" refer to the proposed method obtained by the function `SIMEXBoost` with the argument `Extrapolation = "linear"` and `Extrapolation = "quadratic"`, respectively. L_1-norm and L_2-norm are given by (12). Spe and Sen are specificity and sensitivity, respectively.

p	σ_e	Methods	L_1-norm	L_2-norm	Spe	Sen
200	0.1	Naive	6.900	0.738	0.487	1.000
		SIMEXBoost-Linear	0.116	0.099	1.000	1.000
		SIMEXBoost-Quadratic	0.109	0.090	1.000	1.000
	0.3	Naive	6.350	0.684	0.503	1.000
		SIMEXBoost-Linear	0.101	0.072	1.000	1.000
		SIMEXBoost-Quadratic	0.106	0.075	1.000	1.000
	0.5	Naive	7.600	0.758	0.426	1.000
		SIMEXBoost-Linear	0.257	0.182	1.000	1.000
		SIMEXBoost-Quadratic	0.281	0.188	1.000	1.000
500	0.1	Naive	8.050	0.733	0.732	1.000
		SIMEXBoost-Linear	0.054	0.051	1.000	1.000
		SIMEXBoost-Quadratic	0.069	0.057	1.000	1.000
	0.3	Naive	8.900	0.778	0.710	1.000
		SIMEXBoost-Linear	0.300	0.187	1.000	1.000
		SIMEXBoost-Quadratic	0.300	0.187	1.000	1.000
	0.5	Naive	10.000	0.889	0.692	1.000
		SIMEXBoost-Linear	0.600	0.354	1.000	1.000
		SIMEXBoost-Quadratic	0.600	0.354	1.000	1.000

Table 3: Simulation results for the model (5) with `type="poisson"`. "Naive" is the naive method obtained by the function `Boost_VSE`. "SIMEXBoost-Linear" and "SIMEXBoost-Quadratic" refer to the proposed method obtained by the function `SIMEXBoost` with the argument `Extrapolation = "linear"` and `Extrapolation = "quadratic"`, respectively. L_1-norm and L_2-norm are given by (12). Spe and Sen are specificity and sensitivity, respectively.

p	σ_e	Methods	L_1-norm	L_2-norm	Spe	Sen
200	0.1	Naive	1.600	0.283	0.848	1.000
		SIMEXBoost-Linear	0.208	0.126	1.000	1.000
		SIMEXBoost-Quadratic	0.538	0.407	1.000	1.000
	0.3	Naive	1.750	0.304	0.838	1.000
		SIMEXBoost-Linear	0.178	0.129	1.000	1.000
		SIMEXBoost-Quadratic	0.520	0.301	1.000	1.000
	0.5	Naive	2.500	0.387	0.782	1.000
		SIMEXBoost-Linear	0.366	0.230	1.000	1.000
		SIMEXBoost-Quadratic	1.371	0.877	0.995	1.000
500	0.1	Naive	1.400	0.265	0.944	1.000
		SIMEXBoost-Linear	0.096	0.087	1.000	1.000
		SIMEXBoost-Quadratic	0.428	0.385	1.000	1.000
	0.3	Naive	1.850	0.304	0.930	1.000
		SIMEXBoost-Linear	0.354	0.236	1.000	1.000
		SIMEXBoost-Quadratic	0.279	0.193	1.000	1.000
	0.5	Naive	3.000	0.458	0.903	1.000
		SIMEXBoost-Linear	0.554	0.393	1.000	1.000
		SIMEXBoost-Quadratic	1.238	0.771	1.000	1.000

5.2 Real Data Example

In this section, we apply the package **SIMEXBoost** to analyze the Company Bankruptcy Prediction data, which was from the Taiwan Economic Journal during a period 1999–2009, and it is now available on the Kaggle website (`https://www.kaggle.com/datasets/fedesoriano/company-bankruptcy-prediction`). In this dataset, there are 6819 observations and 95 continuous covariates related to customers' banking records, such as assets, liability, income, and so on. In addition, the response is a binary random variable, where the value 1 reflects that the company is bankrupt, and 0 otherwise. All variables' names and descriptions can be found on the Kaggle website. The main interest in this study is to identify covariates that are informative to the bankruptcy status, and our goal is to apply (3) to characterize the bankruptcy status and covariates.

In addition to detecting important covariates from the multivariate variables, as mentioned in Chen (2023d), measurement error is ubiquitous in variables related to customers' banking records. Hence, one should take measurement error effects into account when doing variable selection. Following the example of the simulation studies, we primarily consider two scenarios: first, without consideration of measurement error, one can directly apply the function `Boost_VSE` to do variable selection. Second, if we wish to implement the function `SIMEXBoost` and take measurement error correction into account, the covariance matrix `sigmae` is needed. Since the dataset has no additional information to estimate $\mathbf{\Sigma}_e$, we may conduct *sensitivity analyses*, which enable us to characterize various degrees of measurement

Table 4: Simulation results for the model (7) with `type="AFT-normal"`. "Naive" is the naive method obtained by the function `Boost_VSE`. "SIMEXBoost-Linear" and "SIMEXBoost-Quadratic" refer to the proposed method obtained by the function `SIMEXBoost` with the argument `Extrapolation = "linear"` and `Extrapolation = "quadratic"`, respectively. L_1-norm and L_2-norm are given by (12). Spe and Sen are specificity and sensitivity, respectively.

p	σ_e	Methods	L_1-norm	L_2-norm	Spe	Sen
200	0.1	Naive	0.875	0.603	1.000	1.000
		SIMEXBoost-Linear	0.378	0.232	1.000	1.000
		SIMEXBoost-Quadratic	0.411	0.258	1.000	1.000
	0.3	Naive	1.450	1.078	1.000	0.667
		SIMEXBoost-Linear	0.400	0.247	1.000	1.000
		SIMEXBoost-Quadratic	0.877	0.605	1.000	1.000
	0.5	Naive	4.700	2.801	1.000	0.667
		SIMEXBoost-Linear	1.614	1.179	0.995	1.000
		SIMEXBoost-Quadratic	2.958	2.204	0.995	1.000
500	0.1	Naive	2.750	1.521	0.998	0.333
		SIMEXBoost-Linear	0.959	0.560	1.000	1.000
		SIMEXBoost-Quadratic	2.710	2.288	0.998	1.000
	0.3	Naive	8.425	5.398	1.000	0.667
		SIMEXBoost-Linear	0.949	0.554	1.000	1.000
		SIMEXBoost-Quadratic	0.820	0.518	1.000	1.000
	0.5	Naive	4.350	2.743	1.000	0.333
		SIMEXBoost-Linear	2.541	1.364	0.998	0.667
		SIMEXBoost-Quadratic	4.003	2.604	0.998	1.000

error and examine the different magnitudes of measurement error effects (e.g., Chen and Yi, 2021; Chen, 2023d). Specifically, we specify $\mathbf{\Sigma}_e$ as a diagonal matrix with common entries being $R = 0.1, 0.3$ and 0.5, and the extrapolation function is taken as linear or quadratic functions in Algorithm 2.

To show the implementation of data analysis, we demonstrate the programming code below. For the convenience of data analysis, we export the full dataset as a csv file, which is available in https://github.com/lchen723/SIMEXBoost.git. One can download the dataset 'bankruptcy_data.csv'. Based on the dataset, we denote `Y` and `Xstar` as the response and the observed covariates, respectively. For the naive method without measurement error correction, we adopt the function `Boost_VSE` with 50 iterations. For the implementation of sensitivity analyses, we use a '`for`' loop for different values of `R`. For each `R`, we run `SIMEXBoost` with `type="binary"` and two extrapolation functions `Extrapolation = "linear"` and `Extrapolation = "quadratic"`.

```
library(MASS)
library(SIMEXBoost)
R = c(0.1, 0.3, 0.5)
data = read.table("bankruptcy_data.csv", sep = ",", head = T)
data = data[, -94]
Y = data[, 95]
Y = as.numeric(Y)
Xstar = t(as.matrix(data[, -95]))
Xstar = scale(Xstar)

p = dim(Xstar)[1]
n = dim(Xstar)[2]

set.seed(202270)
##naive
EST = NULL
naive = Boost_VSE(Y,
                  t(Xstar),
                  type = "binary",
                  Iter = 50,
                  Lambda = 0)$BetaHat
EST = cbind(EST, naive)
##linear
for (i in R) {
  correctL  =  SIMEXBoost(
    Y,
    t(Xstar),
    zeta = c(0, 0.25, 0.5, 0.75, 1),
    B = 50,
    type = "binary",
```

```
    sigmae = diag(i, p),
    Iter = 50,
    Lambda = 0,
    Extrapolation = "linear"
  )$BetaHatCorrect

  EST = rbind(EST, t(correctL))
}
##Quadratic

for (i in R) {
  correctQ  =  SIMEXBoost(
    Y,
    t(Xstar),
    zeta = c(0, 0.25, 0.5, 0.75, 1),
    B = 50,
    type = "binary",
    sigmae = diag(i, p),
    Iter = 50,
    Lambda = 0,
    Extrapolation = "quadratic"
  )$BetaHatCorrect

  EST = rbind(EST, t(correctQ))
}
round(EST, 3)
```

Numerical results are summarized in Table 5, where the column "ID" is the indexes of selected covariates, and the heading "EST" is the estimates of the coefficients. According to our analysis results, we find that the covariate "Net Income to Stockholder's Equity" (ID #90) is only one that is commonly selected from `Boost_VSE` and `SIMEXBoost` under different values of R, which indicates that the covariate #90 is an informative variable regardless of measurement error effects have been corrected or not. For the corrected estimator with different R and extrapolation functions, we observe that variables "Operating Profit Rate: Operating Income/Net Sales" (ID #6), "Pre-tax net Interest Rate: Pre-Tax Income/Net Sales" (ID #7), "Continuous interest rate (after tax): Net Income-Exclude Disposal Gain or Loss/Net Sales" (ID #10), and "Liability-Assets Flag" (ID #85) are selected, except for the scenarios "Correct-L-0.3" and "Correct-L-0.5". Moreover, different variables are selected under different values of R. It might be due to the impact of different magnitudes of measurement error. On the other hand, without measurement error correction, we observe from the naive estimator that most selected variables are different from the proposed estimator, such as "Research and development expense rate: (Research and Development Expenses)/Net Sales" (ID #12), "Tax rate (A): Effective Tax Rate" (ID #15), "Per Share Net profit before tax (Yuan ¥): Pretax Income Per Share" (ID #23), "Total Asset Growth Rate: Total Asset Growth" (ID #29), and "Cash Reinvestment %: Cash Reinvestment Ratio" (ID #32), and those estimates are close to zero. It implies that noninformative variables are possibly selected by the naive method if measurement error is not taken into account in analysis, and it shows an impact of measurement error in data analysis.

Finally, we use the function `proc.time()` to record the system CPU time, and we find that the function `SIMEXBoost` requires 3.84 and 3.50 seconds to run, under `Extrapolation = "linear"` and `Extrapolation = "quadratic"` with a fixed R, respectively, while the function `Boost_VSE` needs 0.45 seconds to derive the estimates. Consistent with the finding in simulation studies, the SIMEXBoost method requires slightly longer computational time than the naive method, which is caused by the repetition of boosting procedure and SIMEX correction.

6 Discussion

The package **SIMEXBoost** provides a novel method for handling high-dimensional data subject to measurement error in covariates. It covers widely used GLM and AFT models in survival analysis, and provides a strategy to deal with variable selection and measurement error correction simultaneously. Moreover, our package is able to handle the collinearity in the covariates. As evidence by longer computational times in numerical studies, the function `SIMEXBoost` seems to be more computationally demanding compared to the naive implementation, which is basically caused by settings of B and $\mathcal{Z}$ for correction of measurement error. Typically, following the similar idea of the Monte Carlo simulation,

Table 5: Variable selection and estimation for the Company Bankruptcy Prediction data based on the model (3) and `type="binary"`. "Naive" refers to the naive method based on `Boost_VSE`. "Correct-L-0.1", "Correct-L-0.3", and "Correct-L-0.5" refer to the function `SIMEXBoost` with `Extrapolation="linear"` and $R = 0.1, 0.3$ and 0.5, respectively. "Correct-Q-0.1", "Correct-Q-0.3", and "Correct-Q-0.5" refer to the function `SIMEXBoost` with `Extrapolation="quadratic"` and $R = 0.1, 0.3$ and 0.5, respectively. The label "−" indicates that the variables are not selected.

ID	EST						
	Naive	Correct-L-0.1	Correct-L-0.3	Correct-L-0.5	Correct-Q-0.1	Correct-Q-0.3	Correct-Q-0.5
6	–	0.400	–	-0.383	3.025	1.781	2.296
7	–	0.474	–	–	2.895	2.126	2.943
8	–	0.322	–	-0.267	3.143	2.628	2.122
10	–	0.396	0.216	–	1.861	1.760	1.749
12	-0.050	–	–	–	–	–	–
15	-0.050	–	–	–	–	–	–
23	0.050	–	–	–	–	–	–
25	–	–	–	–	-0.257	–	–
29	-0.100	–	–	–		–	–
32	-0.050	–	–	–		–	–
37	–	–	-0.235	–	0.766	1.452	0.365
38	-0.150	–	-0.307	–	1.604	1.679	0.404
43	0.050	–	–	–	–	–	–
46	–	-0.300	–	–	0.257	–	–
48	-0.050	–	–	–	–	–	–
55	-0.050	–	–	–	–	–	–
59	0.050	-0.300	–	–	-1.282	-1.285	–
64	-0.050	–	–	–	–	–	–
65	–	0.306	0.300	–	0.916	1.261	–
68	-0.150	–	–	–	–	–	–
74	-0.050	–	–	–	–	–	–
77	-0.050	–	–	–	–	–	–
84	0.050	–	–	–	–	–	–
85	–	-0.472	–	0.237	-6.365	-2.636	-2.956
86	-0.150	–	–	–	–	–	–
87	–	-0.300	–	–	-0.912	–	–
90	-1.350	-1.518	-2.397	-2.433	1.613	-1.602	-3.668
94	0.200	0.240	–	–	-1.553	-1.532	-0.362

larger values of B and $\mathcal{Z}$ usually give the stable estimator but also incur longer computational time. This is a common phenomenon in measurement error analysis and the SIMEX method (e.g., Yi, 2017). In summary, users need to consider whether to take measurement error effects into account based on their data and set arguments, such as B and $\mathcal{Z}$, for the implementation based on their computational resources.

The current state of the package allows for measurement error in continuous covariates and parametric models for exponential family distributed responses or time-to-event outcomes. There are still many possible extensions to the methods, such as consideration of measurement error in binary covariates or measurement error in the response, variable selection for semi-parametric regression models, including the Cox model in survival analysis or the partially linear single index model.

R Software

The R package **SIMEXBoost** is now available on the CRAN website (`https://cran.r-project.org/web/packages/SIMEXBoost/index.html`).

Acknowledgments

The authors would like to thank the editorial team for useful comments to improve the initial manuscript. Chen's research was supported by National Science and Technology Council with grant ID 110-2118-M-004-006-MY2.

References

A. Agresti. *Categorical Data Analysis*. Wiley, New York, 2012. [p8]

E. Alfaro, M. Gamez, L. Garcia, N. Guo, A. Albano, M. Sciandra, and A. Plaia. *adabag: Applies Multiclass AdaBoost.M1, SAMME and Bagging*, 2023. URL https://cran.r-project.org/package=adabag. R package version 5.0. [p6]

K. Bartoszek. *GLSME: Generalized Least Squares with Measurement Error*, 2019. URL https://cran.r-project.org/package=GLSME. R package version 1.0.5. [p6]

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, 2004. [p7]

B. Brown, C. J. Miller, and J. Wolfson. Threeboost: Thresholded boosting for variable selection and prediction via estimating equations. *Journal of Computational and Graphical Statistics*, 26:579–588, 2017. [p5, 7]

R. J. Carroll, D. Ruppert, L. A. Stefanski, and C. M. Crainiceanu. *Measurement Error in Nonlinear Model*. CRC Press, New York, 2006. [p5, 9]

L.-P. Chen. Feature screening based on distance correlation for ultrahigh-dimensional censored data with covariate measurement error. *Computational Statistics*, 36:857–884, 2021. [p5]

L.-P. Chen. BOOME: A python package for handling misclassified disease and ultrahigh-dimensional error-prone gene expression data. *PLOS ONE*, 17:e0276664, 2023a. [p6]

L.-P. Chen. A note of feature screening via rank-based coefficient of correlation. *Biometrical Journal*, 65: 2100373, 2023b. [p5]

L.-P. Chen. De-noising boosting methods for variable selection and estimation subject to error-prone variables. *Statistics and Computing*, 33:38:1–13, 2023c. [p5, 6, 7]

L.-P. Chen. Variable selection and estimation for misclassified binary responses and multivariate error-prone predictors. *Journal of Computational and Graphical Statistics*, 2023d. URL https://doi.org/10.1080/10618600.2023.2218428. [p5, 15, 16]

L.-P. Chen and B. Qiu. Analysis of length-biased and partly interval-censored survival data with mismeasured covariates. *Biometrics*, 79:3929–3940, 2023. [p5, 6, 7, 9]

L.-P. Chen and G. Y. Yi. Analysis of noisy survival data with graphical proportional hazards measurement error models. *Biometrics*, 77:956–969, 2021. [p5, 9, 16]

T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen, R. Mitchell, I. Cano, T. Zhou, M. Li, J. Xie, M. Lin, Y. Geng, Y. Li, J. Yuan, and X. contributors. *xgboost: Extreme Gradient Boosting*, 2023. URL https://cran.r-project.org/package=xgboost. R package version 1.7.5.1. [p6]

Y. Feng, J. Fan, D. F. Saldana, Y. Wu, and R. Samworth. *SIS: Sure Independence Screening*, 2020. URL https://cran.r-project.org/package=SIS. R package version 0.8-8. [p6]

J. Friedman, T. Hastie, R. Tibshirani, B. Narasimhan, K. Tay, N. Simon, J. Qian, and J. Yang. *glmnet: Lasso and Elastic-Net Regularized Generalized Linear Models*, 2023. URL https://cran.r-project.org/package=glmnet. R package version 4.1-7. [p6]

B. Greenwell, B. Boehmke, J. Cunningham, and G. Developers. *gbm: Generalized Boosted Regression Models*, 2022. URL https://cran.r-project.org/package=gbm. R package version 2.1.8.1. [p6]

A. Groll. *GMMBoost: Likelihood-Based Boosting for Generalized Mixed Models*, 2020. URL https://cran.r-project.org/package=GMMBoost. R package version 1.1.3. [p6]

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, 2008. [p5, 6, 7]

B. Hofner, A. Mayr, N. Fenske, J. Thomas, and M. Schmid. *gamboostLSS: Boosting Methods for 'GAMLSS'*, 2023. URL https://cran.r-project.org/package=glmnet. R package version 4.1-7. [p6]

J. F. Lawless. *Statistical Models and Methods for Lifetime Data*. Wiley, New York, 2003. [p5, 8]

W. Lederer, H. Seibold, H. Küchenhoff, C. Lawrence, and R. F. Brøndum. *simex: SIMEX- And MCSIMEX-Algorithm for Measurement Error Models*, 2019. URL https://cran.r-project.org/package=simex. R package version 1.8. [p6]

L. Nab. *mecor: Measurement Error Correction in Linear Models with a Continuous Outcome*, 2021. URL https://cran.r-project.org/package=mecor. R package version 1.0.0. [p6]

B. Qiu and L.-P. Chen. *SIMEXBoost: Boosting Method for High-Dimensional Error-Prone Data*, 2023. URL https://cran.r-project.org/package=SIMEXBoost. R package version 0.2.0. [p6]

Y. Shi, G. Ke, D. Soukhavong, J. Lamb, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.-Y. Liu, N. Titov, Y. Yan, M. Corporation, I. Dropbox, J. Loden, D. Daeschler, G. Rodola, A. Ferreira, D. Lemire, V. Zverovich, I. Corporation, and D. Cortes. *lightgbm: Light Gradient Boosting Machine*, 2023. URL https://cran.r-project.org/package=lightgbm. R package version 3.3.5. [p6]

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of Royal Statistical Society, Series B*, 58:267–288, 1996. [p5]

Z. Wang and T. Hothorn. *bst: Gradient Boosting*, 2023. URL https://cran.r-project.org/package=bst. R package version 0.3-24. [p6]

J. Wolfson. Eeboost: a general method for prediction and variable selection based on estimating equation. *Journal of the American Statistical Association*, 106:296–305, 2011. [p5, 7]

J. Xiong, W. He, and G. Y. Yi. *simexaft: simexaft*, 2019. URL https://cran.r-project.org/package=simexaft. R package version 1.0.7.1. [p6]

G. Y. Yi. *Statistical Analysis with Measurement Error and Misclassication: Strategy, Method and Application*. Springer, New York, 2017. [p18]

Q. Zhang and G. Y. Yi. *augSIMEX: Analysis of Data with Mixed Measurement Error and Misclassification in Covariates*, 2020. URL https://cran.r-project.org/package=augSIMEX. R package version 3.7.4. [p6]

H. Zou. The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101: 1418–1429, 2006. [p5]

H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B*, 67:301–320, 2005. [p5]

Li-Pang Chen
Department of Statistics, National Chengchi University
No. 64, Section 2, Zhinan Rd, Wenshan District, Taipei City, 116
Taiwan (R.O.C.)
ORCiD: 0000-0001-5440-5036
lchen723@nccu.edu.tw

Bangxu Qiu
Department of Statistics, National Chengchi University
No. 64, Section 2, Zhinan Rd, Wenshan District, Taipei City, 116
Taiwan (R.O.C.)
1135427976@qq.com