UiO : Department of Mathematics
University of Oslo

A Comparison of the Shrinkage Effects Between Boosting and Post-Selection Shrinkage Techniques

Wanjuan Ren

Master's Thesis, Spring 2018

This master's thesis is submitted under the master's programme *Modelling and Data Analysis*, with programme option *Statistics and Data Analysis*, at the Department of Mathematics, University of Oslo. The scope of the thesis is 30 credits.

The front page depicts a section of the root system of the exceptional Lie group E_8, projected into the plane. Lie groups were invented by the Norwegian mathematician Sophus Lie (1842–1899) to express symmetries in differential equations and today they play a central role in various parts of mathematics.

Acknowledgments

First and foremost, I would like to thank my supervisor Professor Riccardo De Bin for introducing me to this interesting project, and his enthusiastic guidance and patient supervision throughout the duration of this thesis work.

I would also like to thank my fellow students for our fruitful discussions during the course work.

And last but not least I am very appreciative to the extensive support provided by my family and friends in Norway and China.

Wanjuan Ren
Oslo, Norway
May, 2018

Abstract

Regression analysis is a commonly used approach to modelling the relationships between dependent and independent variables. When estimating the coefficients of a regression model, the least squares estimator is often used, as it has the least variance among the unbiased estimators. When the goal is to make a prediction, nevertheless, one may accept some bias in order to reduce the variance, which may further minimize the prediction error (bias-variance tradeoff). A series of shrinkage methods have therefore been developed to reach the goal. This thesis aims to contrast the shrinkage effects of a selection of methods, such as three post-selection shrinkage methods (global, parameterwise and joint shrinkage), Lasso and boosting. The prediction performances of these methods are compared in the case of only linear effects (case study 1), or a combination of linear and nonlinear effects (case study 2).

In case 1, a simulation study is conducted to compare the prediction performances of different methods under four scenarios. The analysis shows that when the data contains less information (small sample size and large unexplainable variability), the mean squared prediction errors (MSPEs) of the model fitted by Lasso and boosting are smaller than that fitted by the least squares method. The model with parameterwise shrinkage factors (PSF) predicts slightly better than that with global shrinkage factors (GSF). The boosting method with a certain number of iterations calculated by cross-validation produces a model with many variables, whereas with fewer iterations it tends to select only the relevant variables. In case 2, the effects of independent variables are modelled by the fractional polynomial (FP) functions and the prediction performances of different methods, such as multivariable model-building with FP (MFP), FP with shrinkage factors, boosting with FP base-learners, a combination of MFP and Lasso (referred to as MFP-Lasso), and a combination of MFP and boosting (referred to as MFP-boosting), are compared using an artificial (ART) dataset. Small differences in MSPE are found when comparing GSF, PSF and joint shrinkage factors (JSF). Similar results are obtained from the two novel approaches (MFP-boosting and MFP-Lasso). Boosting with the FP base-learners implemented in the R-package mboost has worse prediction performance compared to other methods.

Contents

Chapter 1

Introduction

Regression analysis is a widely used approach to modelling the relationship(s) between a dependent variable and a (set of) independent variable(s). In a linear regression analysis, where the effect of the independent variables is a linear combination of them weighted by the corresponding regression coefficients, the least squares method is often used to estimate the coefficients of a model fitted on a training/learning dataset. The Gauss-Markov theorem shows that in a linear regression model, the least squares estimator is unbiased and optimal in terms of the mean squared error among all unbiased estimators. However, a model with low bias (e.g., linear regression model by least squares) usually has high variance and produces a complex model with a high number of predictors. It tends to fit the training data too well (overfitting) such that the mean squared prediction error (MSPE) of the model fitted on a different dataset is relatively large. A model with extremely large bias and low variance usually produces a simpler model which underfits the data and does not represent the underlying mechanism of data generation. If the objective of an analysis is to make predictions, a balance between these two situations (bias-variance tradeoff) needs to be identified. The variance of the model may be reduced by introducing a low level of bias, thus reducing the expected MSPE. This can be achieved by shrinking the estimates of coefficients towards zero (i.e., shrinkage methods) and/or by forcing some of the coefficients to zero (i.e., variable selection).

In addition, a model with a large number of predictors (e.g., > 10) is difficult to interpret. Hence, identification of a subset of candidate variables which has the strongest effect on the outcome is a commonly used approach. This approach produces a simple model in which the relationships between the response and covariates can be easily interpreted. A simple model can help the researchers to reduce time and costs for collecting the empirical data of interest. For example, in medical research, a relatively accurate prediction model with least covariates can be used to predict the risk of having a certain disease in an unknown patient, thus better utilizing the existing data and reducing the needs for collecting detailed empirical data from every individual.

To shrink the coefficients of a model, a number of techniques have been developed over the past decades. In particular, van Houwelingen and Le Cessie [13] have suggested a simple method which adds a global shrinkage factor (GSF) to the least squares estimates based on

cross-validation. Since not all regression coefficient estimates need to be shrunk towards zero by the same factor, Sauerbrei [16] has further proposed the idea of parameterwise shrinkage factors (PSF), in which a separate shrinkage factor is added to each predictor. The amount of shrinkage is close to zero for predictors with strong effects, whereas the shrinkage can be large for predictors with weak effects. In the cases of dummy variables which represent a category, variables related to combined nonlinear effects, or for variables that are highly correlated, the GSF and PSF may not be good choices, as the effects of such variables should be considered and interpreted jointly. To address these issues, Dunkler and coworkers [6] have proposed a solution of using a common shrinkage factor (joint shrinkage factor/JSF) for the predictors belonging to the same group and associated with each other.

Furthermore, a study by Sauerbrei [16] has suggested that it is not meaningful to apply the shrinkage factor to a full model. On the basis of these advances, van Houwelingen and Sauerbrei [12] have further developed the post-selection shrinkage method which first selects relevant variables using a stepwise selection method (e.g., backward elimination/BE) and then applies the shrinkage factors estimated by cross-validation to the least squares estimates computed on the reduced model. They claimed that the post-selection shrinkage method is equally competitive in prediction and production of sparse models compared to Lasso [12]. To investigate this claim in the present study, the GSF and PSF approaches are contrasted to Lasso and boosting, two of the most popular methods for prediction, in a simple simulation study to evaluate their abilities of prediction, and selection of right variables.

The GSF, PSF and JSF methods are further compared in a complex setting, with the presence of non-linear effects, and their prediction performances are contrasted with that of Lasso and boosting. The choice of modelling non-linear effects through fractional polynomials (FP) allows the assessment of the performance of this specific technique as a base learner in a boosting algorithm. Although this option is implemented in the R-package mboost, to the author's knowledge, no study has investigated its performance. A new approach to using the FP within Lasso (MFP-Lasso) and an alternative application of FP within boosting (MFP-boosting) is also proposed.

Chapter 2

Methods

2.1 Preliminary and notation

A classical linear Gaussian regression model is designed as

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_P x_P + \epsilon,$$

where ϵ is assumed to be independent and to have Gaussian distribution with mean 0 and variance σ^2. The quantities $x_1, \ldots, x_P$ are referred to as predictors (or covariates or independent variables). The quantities $\beta_0, \beta_1, \ldots, \beta_P$ are the coefficients to be estimated on a sample. The random variable Y is called a dependent variable (or response). A regression model can also be written in matrix form $Y = \boldsymbol{x}\boldsymbol{\beta} + \epsilon$, or as $f(\boldsymbol{x}, \boldsymbol{\beta}) + \epsilon$, where $\boldsymbol{x}$ is defined as $\boldsymbol{x} = (1, x_1, \ldots, x_P)$ and $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_P)^T$.

With a given sample $(y_i, x_{i1}, x_{i2}, \ldots, x_{iP})$, $i = 1, \ldots, n$, a standard approach to estimating the coefficients $\boldsymbol{\beta}$ is the ordinary least squares (OLS) method, which minimizes the average squared error

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \boldsymbol{x}_i\boldsymbol{\beta})^2,$$

where $\boldsymbol{x}_i = (1, x_{i1}, \ldots, x_{iP})$. In a matrix form, if the matrix $\boldsymbol{X}$ has full rank, the solution to the problem is

$$\hat{\boldsymbol{\beta}}^{OLS} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{2.1}$$

The least squares method is widely used as it provides an unbiased estimator for the coefficients $\boldsymbol{\beta}$. However, the variance of least squares estimates may be large and may become deficient when the dimension of $\boldsymbol{X}$ is large. Although the least squares method fits the training data well, the prediction performance of the fitted model may be bad on a test dataset. In addition, when the number of variables is larger than 10, the final model given by least squares contains a large number of covariates. As a result, the final model is complicated and difficult to interpret. When building a model, it is often preferred to select covariates which have strong effects on the response of interest [15]. When the goal is to make a prediction, the prediction error can be reduced by introducing some bias in order to reduce the variance.

For a squared loss function, indeed, the prediction error is the sum of the variance of a random error ϵ, the squared bias and the variance [10]. In practice, the prediction error may be reduced by shrinking the regression coefficient estimates towards zero, and/or setting some of them equal to zero. In the rest of the chapter, several methods which deal with variable selection and shrinkage are presented.

2.2 Post-selection shrinkage

One of the methods to improve the prediction accuracy of a model is post-selection shrinkage [12], which mainly consists of two steps. The first step is to choose the covariates to be included in the final model from a large set of candidate variables, for example via BE. Backward elimination is an automated process for sequentially removing the variables which do not have statistical significances to the fit for the final model. Let P denote the total number of candidate covariates (while p is used to denote the number of covariates in the final model). Backward elimination starts with a linear regression model with all covariates and removes the covariates with large p-values until all covariates in the model have p-values smaller than a pre-selected statistical significance level α. Backward elimination can only be applied when the total number of observations is larger than that of the covariates in the full model (i.e., $n > P > p$). The regression coefficients of the reduced model are usually estimated via OLS.

The second step is to apply a shrinkage factor c (estimated by cross-validation) to the least squares estimates to reduce the variance. In the literature, three types of shrinkage factors have been proposed: global [13], parameterwise [16] and joint shrinkage [6], which have been implemented in the present study and are described in detail in the following sections.

2.2.1 Global post-selection shrinkage

The basic principle of global post-selection shrinkage is to add a GSF c to the regression coefficients of the final model (except β_0) [13, 5]. The global post-selection shrinkage method yields the prediction of y

$$\hat{y}_i = \bar{y} + \hat{c} \sum_{j=1}^{p} x_{ij} \hat{\beta}_j^{OLS},$$

where $\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$ and $\hat{c} \in [0, 1]$. The estimates $\hat{\beta}_j^{OLS}, j = 1, \ldots, p$ are the least squares estimates for the model without a shrinkage factor. The GSF c is usually estimated by K-fold cross-validation. The K-fold cross-validation approach randomly divides a dataset into K folds, and utilizes K-1 fold to train the model, and the remaining fold as a test set for model evaluation. When $K = n$, the procedure is referred to as leave-one-out cross-validation. In the present study, the leave-one-out cross-validation method is used to compute c ([6]):

1. For $i = 1, \ldots, n$, compute the least squares estimates of regression coefficients $\hat{\boldsymbol{\gamma}}^{-i}$ on the data leaving x_i and y_i out, where x_i denotes the i-th row vector of the covariate

matrix $\boldsymbol{X}$ and y_i denotes the i-th element of the vector y. The estimator is $\hat{\boldsymbol{\gamma}}^{-i} = (\boldsymbol{X}_{-i}^T \boldsymbol{X}_{-i})^{-1} \boldsymbol{X}_{-i}^T \boldsymbol{y}_{-i}$, where $\boldsymbol{X}_{-i}$ is the covariate matrix without the i-th row vector and $\boldsymbol{y}_{-i}$ is the vector of response without the i-th element.

2. For $i = 1, \ldots, n$, compute the cross-validated linear predictors $\eta_i = x_i \cdot \hat{\boldsymbol{\gamma}}^{-i}$.

3. Compute the least squares estimate of the coefficient of the model with independent variables η_i and dependent variables y_i. The least squares estimate is the global shrinkage estimate. For example, for a linear regression $\hat{y}_i = \hat{\gamma}\eta_i$, $i = 1, \ldots, n$, the global shrinkage estimate $\hat{c} = (\boldsymbol{\eta}^T \boldsymbol{\eta})^{-1} \boldsymbol{\eta}^T \boldsymbol{y}$.

2.2.2 Parameterwise post-selection shrinkage

In most cases, the effect of each predictor on the response can be dissimilar. Therefore, Sauerbrei [16] has suggested to add a different shrinkage factor to the regression coefficient of each predictor. The prediction of the dependent variable, $\hat{y}_i$, is then defined as

$$\hat{y}_i = \bar{y} + \sum_{j=1}^{p} \hat{c}_j x_{ij} \hat{\beta}_j^{OLS},$$

where $\hat{c}_j \in [0, 1]$ for $j = 1, \ldots, p$. The PSF are estimated by the means of leave-one-out cross-validation as described above, with some modifications. The estimation procedure is performed as previously described [6]:

1. For $i = 1, \ldots, n$, compute the least squares estimate of regression coefficients $\hat{\boldsymbol{\gamma}}^{-i} = (\hat{\gamma}_1^{-i}, \ldots, \hat{\gamma}_p^{-i})^T$ for the linear model $\boldsymbol{y}_{-i} = \boldsymbol{X}_{-i} \boldsymbol{\gamma}^{-i} + \boldsymbol{\epsilon}$, i.e., on the data leaving x_i and y_i out.

2. For $i = 1, \ldots, n$ and $j = 1, \ldots, p$, compute the cross-validated linear predictors $\eta_{ij} = x_{ij} \hat{\gamma}_j^{-i}$.

3. Compute the least squares estimate of the coefficients in the same regression model with a covariate matrix $\boldsymbol{H} = \begin{vmatrix} \eta_{11} & \eta_{12} & \ldots & \eta_{1p} \\ \eta_{21} & \eta_{22} & \ldots & \eta_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ \eta_{n1} & \eta_{n2} & \ldots & \eta_{np} \end{vmatrix}$ and dependent variables $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)^T$. The corresponding least squares estimates $(\hat{c}_1, \ldots, \hat{c}_p)$ are the estimates for cross-validation PSF. For example, for a linear regression model $\hat{\boldsymbol{y}} = \boldsymbol{H}\hat{\boldsymbol{c}}$, the PSF are $\hat{\boldsymbol{c}} = (\hat{c}_1, \ldots, \hat{c}_p)^T$.

2.2.3 Joint post-selection shrinkage

Applying a GSF or PSF to some predictors belonging to the same group (e.g., dummy variables or multiple transformations of one predictor) may lead to interpretation problems.

The idea of using joint post-selection shrinkage has been proposed by Dunkler, Sauerbrei and Heinze [6]. Assuming that the set of indices of independent variables is $1, 2, \ldots, p$ and that J_g, $g = 1, \ldots, h$ represents the joined indices that belong to the same group. For example, consider the model with three explanatory variables x_1, x_2, x_3, where two of them x_2 and x_3 belong to the same category, i.e., $J_1 = \{1\}$ and $J_2 = \{2, 3\}$. The prediction of dependent variables is

$$\hat{y}_i = \bar{y} + \sum_{g=1}^{h} \hat{c}_g \sum_{j \in J_g} x_{ij} \hat{\beta}_j^{OLS}.$$

The JSF are estimated by the following procedures [6]:

1. For $i = 1, \ldots, n$, compute the least squares estimate of regression coefficients $\hat{\boldsymbol{\gamma}}^{-i}$ for the linear model $\boldsymbol{y}_{-i} = \boldsymbol{X}_{-i}\boldsymbol{\gamma}^{-i} + \boldsymbol{\epsilon}$, i.e., on the data leaving x_i and y_i out.

2. For $i = 1, \ldots, n$, compute $\eta_{ig} = \sum_{j \in J_g} x_{ij} \hat{\gamma}_j^{-i}, g = 1, \ldots, h$. This yields a cross-validated predictors matrix $\boldsymbol{H} = \begin{vmatrix} \eta_{11} & \eta_{12} & \ldots & \eta_{1h} \\ \eta_{21} & \eta_{22} & \ldots & \eta_{2h} \\ \vdots & \vdots & \vdots & \vdots \\ \eta_{n1} & \eta_{n2} & \ldots & \eta_{nh} \end{vmatrix}$.

3. Compute the least squares estimate of the coefficients in the same regression model with independent variables H and dependent variables $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)^T$. The corresponding least squares estimates $\hat{\mathbf{c}} = (\hat{c}_1, \ldots, \hat{c}_h)^T$ are the estimates for the cross-validation PSF. The shrunk coefficients are $\hat{c}_g \hat{\beta}_j^{OLS}$ for $j \in J_g$. For the example shown above, the shrunk coefficients are obtained as $\hat{c}_1 \hat{\beta}_1^{OLS}$, $\hat{c}_2 \hat{\beta}_2^{OLS}$ and $\hat{c}_2 \hat{\beta}_3^{OLS}$.

2.3 Boosting

Boosting is a widely used algorithm combining a set of results produced by weak learners to a strong learner for improved prediction accuracy. A weak learner is defined as a procedure that predicts slightly better than a random guessing (e.g., coin tossing), whereas a strong learner produces prediction with sufficiently high accuracy. On the basis of these principles, a general algorithm, functional gradient descent (FGD), has been developed by Friedman, Hastie and Tibshirani [8][7] to estimate a real-valued function describing the relationship between an outcome of interest Y and set of predictors $\boldsymbol{X}$ which satisfies

$$f^* = \underset{f}{\operatorname{argmin}} \mathbb{E}[\rho(Y, f(\boldsymbol{X}))],$$

where $\rho(\cdot, \cdot)$ is a loss function assumed to be differentiable with respect to f and convex. Given a training sample of n observations, $\{(y_1, \boldsymbol{x}_1), \ldots, (y_n, \boldsymbol{x}_n)\}$, boosting estimates the optimal function by minimizing the average sum of the empirical risk $\frac{1}{n} \sum_{i=1}^{n} \rho(y_i, f(\boldsymbol{x}_i))$. Later, Buhlmann and Yu [4] have investigated the algorithm focusing on the specific loss

function $\frac{1}{2}(y-f)^2$. The algorithm is referred to as L_2 boosting. The L_2 boosting method can be applied to estimate the coefficients of a linear regression model as shown in the box below [3].

L_2 boosting algorithm

1. Initialize the function estimate $\hat{f}^{[0]}$ with offset values. The default choice is

$$\hat{f}^{[0]} = (\bar{y}, \ldots, \bar{y})^T,$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$.

2. For $m = 1, \ldots, m_{stop}$:

   (a) Compute the negative gradient vector

$$\boldsymbol{U} = (u_1, \ldots, u_n)^T = (y_1 - \hat{f}^{[m-1]}(\boldsymbol{x}_1), \ldots, y_n - \hat{f}^{[m-1]}(\boldsymbol{x}_n))^T.$$

   (b) Fit a base-learner $g(\cdot)$ (e.g., linear regression) to the negative gradient vector and obtain the estimates $\hat{g}^{[m]}(\cdot)$ (e.g., $g^{[m]} = \boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{U}$).

   (c) Update the function estimates

$$\hat{f}^{[m]}(\cdot) = \hat{f}^{[m-1]}(\cdot) + \nu \cdot \hat{g}^{[m]}(\cdot),$$

   where $\nu \in (0, 1]$ is a tuning parameter.

In step 2(b), a base-learner with all dimensions of X (predictors) is fitted and added to the update of the function estimates. Therefore, the final model includes all predictors. In order to extend the procedure to high-dimensional data, where the number of predictors is higher than the observations ($P > n$), and to perform an automated variable selection in the fitting process, a component-wise version of the gradient boosting has been proposed [1, 2, 8, 7]. The detailed algorithm is illustrated in the box below. The component-wise L_2 boosting fits a (or a set of) base-learner(s) of each predictor (e.g., $g_1(\cdot), \ldots, g_p(\cdot)$) separately to the negative gradient vector in step 2(b) and selects the best fit with least sum of squared residuals in each iteration. The type of base-learner determines the way of entering into the final model for each variable. For example, the base-learners of a linear regression model $f(x) = \beta_0 + \beta_1 x_1 + \cdots + \beta_P x_P$ are a series of linear functions $g_1(x_1) = \beta_1 x_1, \ldots, g_p(x_P) = \beta_P x_P$. The coefficients are estimated by the least squares method. In each iteration, the base-learners are fitted separately to the negative gradient vector for each dimension of $\boldsymbol{X}$ and the best-performance base-learner (that reduces most of the loss function) is selected and added to update the function estimate. Therefore, predictors with strong effects on the outcome are likely selected in the function estimate in the first iterations. Those with weak or none effects are excluded if the algorithm stops early enough. For $m_{stop} \to \infty$, the final model with boosting converges to the least squares model in the case of $n < P$.

Component-wise L_2 boosting algorithm

1. Initialize the function estimate $\hat{f}^{[0]}$ with offset values. The default choice is

$$\hat{f}^{[0]} = (\bar{y}, \ldots, \bar{y})^T,$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$.

2. Specify a set of base-learners $g_1(\cdot), \ldots, g_j(\cdot)$ with respect to the covariates $\boldsymbol{x}_{\cdot j} = (x_{1j}, \ldots, x_{nj})^T$, $j = 1, \ldots, P$.

3. For $m = 1, \ldots, m_{stop}$:

   (a) Compute the negative gradient vector

$$\boldsymbol{U} = (u_1, \ldots, u_n)^T = (y_1 - \hat{f}^{[m-1]}(\boldsymbol{x}_1), \ldots, y_n - \hat{f}^{[m-1]}(\boldsymbol{x}_n))^T.$$

   (b) Fit base-learner of each predictor to the negative gradient vector $\boldsymbol{U}$. The corresponding estimates are $\hat{g}_j^{[m]}(\boldsymbol{x}_{\cdot j})$, $j = 1, \ldots, P$.

   (c) Select the best-performance base-learner $\hat{g}_{j^*}(\cdot)$ with least sum of squared residuals where

$$j^* = \min_j \sum_{i=1}^{n} (u_i - \hat{g}_j^{[m]}(x_{ij}))^2.$$

   (d) Update the function estimates

$$\hat{f}^{[m]}(\cdot) = \hat{f}^{[m-1]}(\cdot) + \nu \cdot \hat{g}_{j^*}^{[m]}(\cdot).$$

In the boosting algorithm, the number of iterations m_{stop} and the step size ν are tuning parameters. If ν is close to 1, the total effects of base-learners are included in the function estimate of each iteration and boosting estimates converge to the least squares estimates with very few steps (making it unlikely to find the right amount of shrinkage). When m_{stop} is too large and ν is small, the results of both classical L_2 boosting and its component-wise version tend to fit the training data too well and to converge to the least squares estimates (overfitting). With small values of m_{stop} and ν, L_2 boosting tends to underestimate the coefficients for covariates and component-wise boosting tends to produce a sparse model which may not contain all relevant variables (underfitting). This results in not only bad estimation but also poor prediction performance. Therefore, a common approach in practice is to fix ν small (e.g., $\nu = 0.1$, see [3]) and find the optimal value of m_{stop} which generates a best-performance model according to the purpose of analysis (e.g., prediction). The prevalent methods to estimate m_{stop} are information criteria (e.g., AIC) and resampling approaches (e.g., cross-validation). In this study, a 10-fold cross-validation is used to select the tuning parameter m_{stop}. Cross-validation divides the data into training and test sets and chooses

the optimal value of m_{stop} to minimize the average empirical loss of the out-of-sample test data. The loss function in the cross-validation procedure needs to be consistent with that applied in the boosting algorithm. For example, the optimal m_{stop} by cross-validation for L_2 boosting minimizes the sum of squared error of the test data.

2.4 Lasso

Lasso is a shrinkage method mainly used for prediction. This method fits a regression model with L_1 regularization, which performs shrinkage and variable selection. Basically, Lasso adds a penalty term (L_1 penalty) to the squared loss to prevent overfitting.

Consider a linear regression model

$$f(x) = \beta_0 + \sum_{j=1}^{P} x_j \beta_j,$$

the least squares estimates are given by minimizing the sum of squared residuals

$$\sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{P} x_{ij}\beta_j)^2.$$

The Lasso method introduces a penalty term and minimizes the penalized sum of squared residuals

$$\sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{P} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{P} |\beta_j|, \tag{2.2}$$

where λ needs to be nonnegative and controls the amount of shrinkage. With $\lambda = 0$ (no shrinkage at all), the Lasso estimates are equal to the least squares estimates. When λ increases, the level of shrinkage also increases accordingly, and the Lasso estimates get closer to zero. Some of these with the weakest effects are forced to be exactly zero (variable selection). The term $\sum_{j=1}^{P} |\beta_j|$ is the L_1 norm, therefore it is referred to as L_1 penalty. The penalized sum of squared residuals of Lasso can also be written as following:

$$\begin{aligned} &\sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{P} x_{ij}\beta_j)^2, \\ &\text{subject to} \sum_{j=1}^{P} |\beta_j| \leq t, \end{aligned} \tag{2.3}$$

where t is the parameter that controls the amount of shrinkage. The parameter λ and t are in one-to-one relationship. The level of shrinkage depends on t (smaller t, more shrinkage). When t is sufficiently small, the Lasso estimates for some covariates are shrunk to zero. Therefore, the choice of t can influence the number of independent variables in the final model.

There is currently no closed form solution to this problem and the estimates of the regression coefficients need to be computed numerically. Clever algorithms, for example coordinate descent [9], have been implemented.

The K-fold cross-validation method is typically used to select the optimal value of the tuning parameter t or λ. The number of folds is normally 5 or 10. The computing time increases with the number of folds. The procedure of K-fold cross-validation for choosing the optimal tuning parameter λ is described in details below. The procedure for the choice of t is similar to that for the choice of λ.

K-fold Cross-validation for Lasso

1. Randomly divide the dataset $(\boldsymbol{x}_i, y_i), i = 1, \ldots, n$ into K folds $F_k, k = 1, \ldots, K$ with approximately equal size. $K = 5$ or 10 is a common choice.
2. Determine a discrete set $\Lambda = \{\lambda_1, \ldots, \lambda_M\}$ containing possible values for the tuning parameter λ.
3. For $k = 1, \ldots, K$:
   (a) use $(x_i, y_i), i \in F_k$ for test data and $(\mathbf{x}_i, y_i), i \notin F_k$ for training data,
   (b) standardize the training data such that each column has mean 0 and variance 1,
   (c) for $m = 1, \ldots, M$:
      i. compute the Lasso estimates on the training data for each value in Λ and predict the outcome in the (standardized) test data $\hat{f}^{-k}_{\lambda_m}(x_i), i \in F_k$,
      ii. compute the sum of squared test error for each value in Λ

$$e_k(\lambda_m) = \sum_{i \in F_k} (y_i - \hat{f}^{-k}_{\lambda_m}(x_i))^2.$$

4. For each value of λ in the set Λ, compute the average test error over all folds

$$CV(\lambda) = \frac{1}{K} \sum_{k=1}^{K} e_k(\lambda) = \frac{1}{n} \sum_{k=1}^{K} \sum_{i \in F_k} (y_i - \hat{f}^{-k}_{\lambda}(x_i))^2.$$

5. Choose the λ which minimizes $CV(\lambda)$.

2.5 Fractional polynomials

The relationship between an independent variable x and a response y is often complex and nonlinear. In this case, it is inappropriate to apply a linear regression to the modelling.

For example, when nonlinearity is not taken into account, a predictor with nonlinear effect is likely excluded in the process of model building when using one of the variable selection methods [14]. For this reason, the FP method has first been introduced by Royston and Altman [14] and further modified by Sauerbrei and Royston [17] for modelling complicated relationships including linear and nonlinear effects. In this section, a brief introduction to FP for single covariate and multiple covariates is given. An algorithm for building a multivariable FP model is also described.

2.5.1 Univariable fractional polynomial models

An *FP model of degree 1* (an FP1 model) for a positive continuous variable is defined as

$$f(x) = \beta_0 + \beta_1 g(x, q),$$

where $g(x, q) = x^q$ and power q is selected from a predefined set $\mathcal{S} = \{-2, -1, -0.5, 0, 0.5, 1, 2, 3\}$. The function $g(x, q) = x^q$ is referred to as *an FP function of degree 1*. With $q = 0$, x^0 is defined to be $\log(x)$ by convention. When $q = 1$, the FP1 model corresponds to a linear model $f(x) = \beta_0 + \beta_1 x$.

To model more irregular relationships between y and x, a polynomial of degree 1 may be insufficient. Accordingly, Royston and Altman [14] have also proposed an *FP model of degree 2* (FP2) which is

$$f(x) = \beta_0 + \beta_1 g(x, q_1) + \beta_2 g(x, q_2),$$

where $q_1 \in \mathcal{S}$ and $q_2 \in \mathcal{S}$. When $q_1 = q_2$, $g(x, q_1) = x^{q_1}$ and $g(x, q_2) = \log(x)x^{q_2}$. When $q_1 \neq q_2$, $g(x, q_1) = x^{q_1}$ and $g(x, q_2) = x^{q_2}$. For example, an FP2 model with powers (2, 2) (i.e., $q_1 = q_2 = 2$) is $f(x) = \beta_0 + \beta_1 x^2 + \beta_2 \log(x)x^2$. An FP2 model with powers (-1, 0) is $f(x) = \beta_0 + \beta_1 x^{-1} + \beta_2 \log(x)$. The family of FP2 provides 36 various combinations of powers q_1 and q_2 including 28 of case $q_1 \neq q_2$ and 8 of case $q_1 = q_2$. Therefore an FP2 model provides more varieties of functional forms than an FP1 model which only contains 8 types of functions.

In general, a d-degree FP function (model) is defined as

$$f(x) = \beta_0 + \sum_{j=1}^{d} \beta_j g(x, q_j),$$

where d denotes the degree of the FP function and $q_j \in \mathcal{S}$. The degree of an FP function is possible to be larger than two, but the flexibility of an FP2 function is in fact enough for most cases [15]. In addition, a model with an FP function of degree larger than two may be too complicated to interpret and the fitted model may overfit the training data. Therefore, in the following analysis, the maximal permitted degree of 2 is considered for a FP function.

The FP function can be applied to regression models with Gaussian random error, generalized linear models, generalized additive models and survival models. In this study, only the Gaussian errors are considered. In addition, the coefficients of the model are estimated by maximizing the log-likelihood function with respect to each coefficient.

2.5.2 Multivariable fractional polynomial models

Analysis of real-life datasets involves modelling the effect of multiple covariates on an outcome. The idea of the FP function described above is extended to multiple covariates. For completeness, the presence of categorical covariates is also considered in the model. Although the maximal permitted degree for FP can be more than two as previously introduced, only two is considered in the present work. Let h be the number of continuous variables and r be the number of binary variables. An FP model is defined as

$$f(x) = \beta_0 + \sum_{j=1}^{h} \sum_{k=1}^{d_j} \beta_{jk} g(x_j, q_{jk}) + \sum_{j=h+1}^{h+r} \beta_j x_j,$$

where $d_j \in \{1, 2\}$ denotes the degree of a multivariable FP function for covariate x_j and all powers belong to the set $\mathcal{S}$. When $d_j = 2$,

$$\begin{cases} g(x_j, q_{j1}) = x_j^{q_{j1}} \text{ and } g(x_j, q_{j2}) = x_j^{q_{j2}} \log(x) & \text{if } q_{j1} = q_{j2}, \\ g(x_j, q_{j1}) = x_j^{q_{j1}} \text{ and } g(x_j, q_{j2}) = x_j^{q_{j2}} & \text{if } q_{j1} \neq q_{j2}. \end{cases}$$

For example, a multivariable FP model with two continuous variables and one binary variables is $f(x) = \beta_0 + \sum_{j=1}^{2} \sum_{k=1}^{d_j} \beta_{jk} g(x_j, q_{jk}) + \beta_3 x_3$. If $d_1 = 1$ and $d_2 = 2$, it becomes $f(x) = \beta_0 + \beta_{11} g(x, q_{11}) + \beta_{21} g(x, q_{21}) + \beta_{22} g(x, q_{22}) + \beta_3 x_3$.

When building a multivariable FP model for a given training dataset, it is crucial to choose the FP forms and select the covariates with influences on the outcome. Sauerbrei and Royston [17] have introduced an algorithm which is referred to as multivariable model-building with FP (MFP). The MFP algorithm runs for several cycles. In each cycle, a variable selection method, BE, is applied to choose continuous variables and binary variables which are significant to the outcome Y at the level α_1. To determine the optimal FP transformation for each continuous variable, MFP first picks out the best-performance functions for both degree 1 and degree 2 (best FP1 and best FP2), which have the largest log-likelihood among all possible functions. Next, it finds the best function for the variable by the log-likelihood ratio (χ^2) test at a significance level α_2. The test statistic is minus twice the difference in the log-likelihood. A typical choice for the significance levels is $\alpha_1 = \alpha_2 = 0.05$. Normally, the algorithm converges after 2-4 cycles [17]. The MFP algorithm is described as following [15]:

1. Choose the significance level α_1 for BE and α_2 for the log-likelihood ratio test.
2. Choose the maximum permitted degree d for the FP functions. The default choice is $d = 2$.
3. Fit the full linear model, which is $f(x) = \beta_0 + \sum_{j=1}^{h+r} \beta_j x_j$. Initialize the final model $M_{final} = \beta_0 + \sum_{j=1}^{h+r} \beta_j x_j$.
4. Order the variables increasingly according to their p-values. The variables are now $x_{(1)}, x_{(2)}, \ldots, x_{(h+r)}$ and the final model is $M_{final} = \beta_0 + \sum_{j=1}^{h+r} \beta_j x_{(j)}$.

5. Initialize the cycle $c = 0$.

6. For $j = 1, \ldots, h + r$:

   (a) If $x_{(j)}$ is binary, performs a log-likelihood ratio test at the significance level α_1 of M_{final} against the same model without $x_{(j)}$. If $x_{(j)}$ is significant, keep it in the model M_{final}. Otherwise, remove it.

   (b) If $x_{(j)}$ is continuous:

      i. **Define $\mathbf{M_{final}}$ and $\mathbf{M_1}$:** $M_{final} = M_1 + \beta_j x_{(j)}$, where M_1 contains all the covariates in M_{final} except $x_{(j)}$. If the variable $x_{(j)}$ is omitted from M_{final} in the last cycle, $\beta_j = 0$.

      ii. **Search for the best FP1:** compute the log-likelihood of 8 models $M_1 + \beta_j x^{q_j}$, where $q_j \in \mathcal{S}$ and choose the best-performance FP1 model with the largest log-likelihood (best FP1 for $x_{(j)}$).

      iii. **Search for the best FP2:** compute the log-likelihood of 36 models $M_1 + \beta_{j1} g(x_{(j)}, q_{j1}) + \beta_{j2} g(x_{(j)}, q_{j2})$, where $q_{j1}, q_{j2} \in \mathcal{S}$ and choose the FP2 model with the largest log-likelihood (best FP2 for $x_{(j)}$).

      iv. **Test best FP2 vs $\mathbf{M_1}$:** perform a χ^2 test of the best FP2 against the model M_1 (i.e., $x_{(j)}$ excluded) at the significance level α_1. If the test is significant, continue. Otherwise, update $M_{final} = M_1$.

      v. **Test FP2 vs linear:** perform a χ^2 test of the best 2-degree FP model against $M_1 + \beta_j x_{(j)}$ at the significance level α_2. If the test is significant, continue. Otherwise, update the $M_{final} = M_1 + \beta_j x_{(j)}$.

      vi. **Test FP2 vs FP1:** perform a χ^2 test of the best 2-degree FP model against the best FP1 model at the significance level α_2. If the test is significant, update M_{final} as the best FP2 model. Otherwise, update M_{final} as the best FP1 model.

7. Update the cycle counter $c = c + 1$. If the model M_{final} in cycle $c - 1$ is the same as in cycle c, the algorithm converges. Stop the procedure and return the fitted FP model. If $c > c_{max}$ where c_{max} is the maximum permitted cycles, stop the procedure and return that the algorithm failed to converge in c_{max} cycles.

2.6 Fractional polynomials with shrinkage

The MFP algorithm utilizes a stepwise variable selection method which either includes or omits a covariate in each step until all covariates in the model are significant at level α_1. To improve the prediction ability of the model given by MFP, it is possible to exploit the idea of the post-selection shrinkage methods. Briefly, the MFP algorithm is run on a set of training data for variable selection and for identification of the FP function. The resulting model by MFP is referred to as M_{final}. Afterwards, it introduces a shrinkage factor(s), which may be

GSF, PSF or JSF, to the model M_{final}. For example, if the final model identified by the MFP algorithm is $f(x) = \hat{\beta}_0 + \hat{\beta}_1 x_1^2 + \hat{\beta}_{21} x_2^{-0.5} + \hat{\beta}_{22} x_2^2 + \hat{\beta}_3 x_3$,

1. the model with a GSF is

$$f(x) = \hat{\beta}_0 + c\hat{\beta}_1 x_1^2 + c\hat{\beta}_{21} x_2^{-0.5} + c\hat{\beta}_{22} x_2^2 + c\hat{\beta}_3 x_3.$$

2. the model with PSF is

$$f(x) = \hat{\beta}_0 + c_1\hat{\beta}_1 x_1^2 + c_2\hat{\beta}_{21} x_2^{-0.5} + c_3\hat{\beta}_{22} x_2^2 + c_4\hat{\beta}_3 x_3.$$

3. the model with JSF is

$$f(x) = \hat{\beta}_0 + c_1\hat{\beta}_1 x_1^2 + c_2\hat{\beta}_{21} x_2^{-0.5} + c_2\hat{\beta}_{22} x_2^2 + c_3\hat{\beta}_3 x_3.$$

The shrinkage factor is estimated by cross-validation as described in Section 2.2. The predictive power of the methods is investigated with an artificial dataset created by simulation from a real data study in Chapter 4.

2.7 Fractional polynomials and boosting

2.7.1 Fractional polynomials as base-learners

As described in Section 2.3, component-wise boosting performs variable selection in the fitting process and produces shrunk estimates of the regression coefficients. A set of base-learners are fitted on the negative gradient to iteratively improve the model. One possible choice for the base-learners is a class of FP1 and FP2. Implementation of this approach using the R-package mboost [11] is presented herein. For each positive continuous variable x, there are sixteen base-learners which are $\beta_0+\beta_1 x^{-2}, \beta_0+\beta_1 x^{-1}, \ldots, \beta_0+\beta_1 x^2, \beta_0+\beta_1 x^3, \beta_0+\beta_1 x^{-2}\log(x), \ldots, \beta_0+\beta_1 x^3\log(x)$. A base-learner for a binary variable x is $\beta_0 + \beta_1 x$. In the case of a model with two continuous variables x_1, x_2 and one binary variable x_3, there are sixteen base-learners for x_1, sixteen for x_2 and one for x_3 in the boosting algorithm. Each of them is fitted to the negative gradient vector (see the component-wise L_2 boosting algorithm in Section 2.3) and the one with the least mean squared error is selected and added to update the function estimate. This procedure is repeated at each boosting iteration.

The final model given by boosting is different from the FP model defined in Section 2.5 as the structure of each continuous variable in the final model is an additive combination of functions selected from sixteen base-learners. At each iteration, indeed, a different form of FP may be selected, and the resulting function is often a mixture of these. For example, in the case of a regression analysis with only one covariate, boosting with FP can produce a final model after M iterations, $f(x) = \beta_0 + \beta_1 x^{-2} + \beta_2 x^{-0.5} + \beta_3 x^2\log(x) + \beta_4\log(x)^2$. Component boosting with FP transformation provides more varieties of functional forms compared to the FP2 model. However, with many steps, boosting may yield an extremely

complicated model and result in overfitting. It is therefore crucial to stop the algorithm before convergence and keep the step size ν small (e.g., $\nu = 0.1$). The number of step for stopping the procedure m_{stop} may also be estimated by AIC and the resampling method. In the R-package mboost, cross-validation is served as a default choice. This method will be investigated in Chapter 4 by fitting a set of data simulated from a real-life study.

2.7.2 Boosting on a fractional polynomials model

As the method introduced in the previous section produces a model which does not have the same form as the FP model as described in Section 2.5, an alternative approach is here proposed, which uses the MFP algorithm to determine the FP functional form of continuous variables and the component-wise L_2 boosting to fit the model and perform variable selection. In essence, the procedure contains two steps. In the first step, the MFP algorithm is applied to select the FP function for continuous variables. In the second step, boosting is performed with the linear base-learners for the binary variables and the continuous variables which are not selected by MFP and the fixed base-learners selected by MFP for the variables with nonlinear effect. For example, in the case of three candidate variables x_1, x_2, x_3, of which x_1 and x_2 are continuous and x_3 is binary. The MFP algorithm selects the FP function $log(x_1)$ for x_1. The rest of the variables are not selected by MFP. The base-learners for the component-wise L_2 boosting is therefore $\beta_1 log(x_1)$, $\beta_2 x_2$ and $\beta_3 x_3$. The prediction performance of this method is evaluated based on MSPE on a set of test data in later analysis.

2.8 MFP and Lasso

The coefficients of variable in the MFP algorithm is estimated by least squares. Instead of introducing shrinkage factors to the model given by the MFP algorithm as described in Section 2.6, a novel method which combines MFP with Lasso (MFP-Lasso) is proposed. The idea is to take the advantages of Lasso (shrinkage, variable selection) once the effect of the predictors are modelled by FP. A two-step algorithm is proposed: (i) the MFP algorithm is applied to select the FP function for each continuous covariate, (ii) Lasso is applied to estimate the coefficients and perform variable selection. The covariates which are eliminated by MFP enter into the model in linear forms in Lasso. The procedure is described as following:

1. Run the MFP algorithm on a training dataset and obtain the model M_{final}.
2. Build a model M_2 which contains all terms in model M_{final} and the linear terms of the covariates omitted by MFP.
3. Standardize the transformed predictors in the training dataset.
4. Run K-fold cross-validation for Lasso (see Section 2.4) to achieve the optimal value of tuning parameter.

5. Fit the final model M_2 using Lasso with the optimal value for the tuning parameter based on cross-validation.

Chapter 3

Small simulation study

In this chapter, a set of training data and test data is simulated according to the method suggested by van Houwelingen and Sauerbrei [12]. A linear regression model with Gaussian distribution is fitted using the following methods: least squares, post-selection shrinkage, L_2 boosting and Lasso. The fitted models are evaluated for the prediction performances based on the MSPEs. One of the main goals is to add boosting to the comparison.

3.1 Design of the simulation data

In the simulation data, fifteen covariates $x_1, \ldots, x_{15}$ are generated (i.e., P=15). The covariates have multivariate normal distribution with mean $\boldsymbol{\mu} = (0, \ldots, 0)^T$ and covariance matrix

$$\boldsymbol{\Sigma} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0.7 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & -0.7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.7 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.5 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 \\
0 & 0 & 0 & -0.7 & 0 & 0 & 0.3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0.5 & 0 & 0 \\
0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.7 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}.$$

The regression coefficients for generating the response y are

$$\boldsymbol{\beta} = (0, 0, 0, -0.5, 0.5, 0.5, 0.5, 1, 1, 1.5, 0, 0, 0, 0, 0)^T.$$

The coefficients of variables $x_4, \ldots, x_{10}$ are nonzero, indicating that they are included in the model. Most of the variables are correlated with each other (Figure 3.1). The correlation between the variables are $\rho_{x_1,x_5} = 0.7, \rho_{x_1,x_{10}} = 0.5, \rho_{x_2,x_6} = 0.5, \rho_{x_4,x_8} = -0.7, \rho_{x_8,x_7} = 0.3, \rho_{x_7,x_{14}} = 0.5, \rho_{x_9,x_{13}} = 0.5, \rho_{x_{11},x_{12}} = 0.7$. The variables x_3 and x_{15} are independent which are not correlated with any other variables. The response y is generated by adding a random

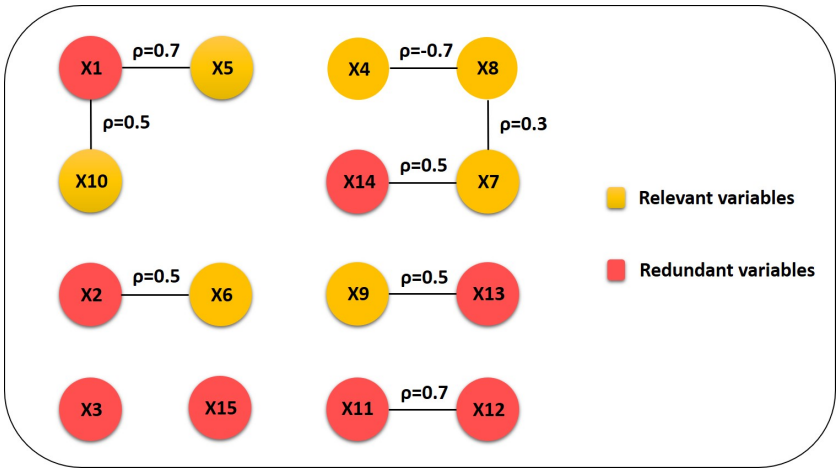

Figure 3.1: An illustration of the relationships between different variables. Variables in yellow are relevant for the true model; Variables in pink are redundant for the true model. ρ denotes the correlation value.

error ϵ with Gaussian distribution with mean 0 and variance σ^2. That is

$$y = \boldsymbol{x}\boldsymbol{\beta} + \epsilon,$$

where $\boldsymbol{x} = (x_1, \ldots, x_{15})$ and $\epsilon \sim N(0, \sigma^2)$. In the study of van Houwelingen and Sauerbrei [12], four scenarios are constructed with combinations of two sample sizes n and two variances of random error ϵ as illustrated in Table 3.1. In this study $N = 2{,}000$ samples are generated, meaning that the whole fitting process using different methods is repeated 2,000 times. In addition to the training data, the test data is also generated using the same procedure to compute the MSPE.

Table 3.1: Sample sizes and random errors of each scenario

Scenario number	Sample size n	Random error ϵ
1	100	$\epsilon \sim N(0, 6.25)$
2	100	$\epsilon \sim N(0, 2.5)$
3	400	$\epsilon \sim N(0, 6.25)$
4	400	$\epsilon \sim N(0, 2.5)$

3.2 Model fitting

A linear regression model is built using the training data. Consider a regression model

$$y = \sum_{j=1}^{15} x_j \beta_j + \epsilon,$$

where $\epsilon \sim N(0, \sigma^2)$ and σ is unknown. For each scenario, the model is fitted via the following methods:

Full:	a full model estimated by least squares with all variables.
Full with GSF:	a full model with a GSF (based on cross-validation).
BE(0.01):	BE with the significance level $\alpha = 0.01$.
BE(0.01) with GSF:	BE with GSF with the significance level $\alpha = 0.01$.
BE(0.01) with PSF:	BE with PSF with the significance level $\alpha = 0.01$.
BE(0.05):	BE with the significance level $\alpha = 0.05$.
BE(0.05) with GSF:	BE with GSF with the significance level $\alpha = 0.05$.
BE(0.05) with PSF:	BE with PSF with the significance level $\alpha = 0.05$.
BE(0.157):	BE with the significance level $\alpha = 0.157$ (corresponding to AIC [15]).
BE(0.157) with GSF:	BE with GSF with the significance level $\alpha = 0.157$.
BE(0.157) with PSF:	BE with PSF with the significance level $\alpha = 0.157$.
Lasso:	Lasso with the tuning parameter λ based on the 10-fold cross-validation (see Section 2.4).
Boosting:	L_2 component-wise boosting with the linear base-learner βx for each covariate (see Section 2.3) and the maximum number of iterations 500.

The whole procedure is described in detail as following:

1. For scenario $1, \ldots, 4$:
2. For $i = 1, \ldots, N$:
   (a) **Data generation:** generate n number of training observations and n number of test observations according to the setting in Section 3.1.

(b) **Model fitting:** fit the model by the methods described above and 13 fit of model are obtained.

(c) **Prediction:** compute the prediction of the 13 fitted models on test data $\hat{y}_1, \ldots, \hat{y}_n$.

(d) **Prediction error:** compute the MSPE, $MSPE_i = \frac{1}{n}\sum_{k=1}^{n}(y_k - \hat{y}_k)^2$.

3. Compute the average MSPE over $N = 2{,}000$ repetitions which is $\frac{1}{N}\sum_{i=1}^{N} MSPE_i$.

3.3 Results and discussions

3.3.1 Prediction performance

Table 3.2 illustrates the average of the MSPEs of the models fitted by the methods described above over 2,000 repetitions under each scenario. The results are in agreement with the previous findings by van Houwelingen and Sauerbrei [12], thus verifying the soundness of the present approach.

It is apparent that the average MSPE under scenario 3 and 4 are smaller than that under scenario 1 and 2, respectively, clearly indicating that the learning methods normally have better prediction performance when the sample size is larger.

Boosting yields similar results as that produced by Lasso. The average MSPE is much smaller than least squares under scenario 1 and 2, whereas the prediction performance of boosting is close to least squares under scenario 3 and 4. Boosting and Lasso have reasonable performances in the case of small sample size. For scenario 1 and 2, these methods have smaller average MSPE than least squares. However, when the sample size increases, both methods tend to have performances similar to the full model with the least squares estimators. For all four scenarios, boosting and Lasso yield prediction models slightly better than the full model. It may be concluded that for the purpose of prediction, boosting and Lasso are both reasonable choices.

The averaged MSPEs of the post-selection shrinkage methods are smaller than that of the BE method with the same significance level. It is likely that applying the shrinkage factors to the models selected by BE may influence the prediction performances. The choice of the significance level also has influence on the prediction performance. In the case of the significance level $\alpha = 0.57$ and $\alpha = 0.05$, the resulting models under all scenarios predict better than least squares. In the case of $\alpha = 0.01$, however, the prediction performance tends to vary dramatically. With a small sample size ($n = 100$), the resulting models by BE and post-selection shrinkage are worse than the full model in terms of prediction accuracy. When the sample size is relatively large ($n = 400$), BE and post-selection shrinkage tend to have similar performances as other methods. Under scenario 4, BE with GSF at the significance level $\alpha = 0.01$ yields the best prediction among all methods.

Figure 3.2-3.5 illustrates the distribution of the MSPEs of the fitted models over 2,000 repetitions under each scenario. Figure 3.6-3.9 illustrates the differences in MSPE between all fitted and full models with least squares estimators under each scenario, in which the MSPE of the full model is used as a baseline. While Table 3.2 confirms the results of

van Houwelingen and Sauerbrei [12], which shows a slightly better performance of the PSF than GSF or Lasso when MSPEs are averaged, Figure 3.2-3.5 and 3.6-3.9 show a noticeable variability (not shown in the original study) among the 2,000 repetitions, which reveals that the differences in the performance of the methods are rather minimal.

Table 3.2: The average of the mean squared prediction errors (MSPEs) of the models fitted by different methods over 2,000 repetitions. Full: least squares; GSF: global shrinkage factors; BE: backward elimination; PSF: parameterwise shrinkage factors.

Method (significance level)	Scenario 1	Scenario 2	Scenario 3	Scenario 4
Full	7.360	2.944	6.502	2.599
Full with GSF	7.230	2.922	6.495	2.598
BE (0.01)	7.459	3.034	6.498	2.556
BE (0.01) with GSF	7.419	3.024	6.496	2.555
BE (0.01) with PSF	7.414	3.019	6.496	2.557
BE (0.05)	7.316	2.922	6.455	2.563
BE (0.05) with GSF	7.259	2.911	6.452	2.562
BE (0.05) with PSF	7.227	2.898	6.450	2.561
BE (0.157)	7.308	2.907	6.466	2.577
BE (0.157) with GSF	7.222	2.892	6.462	2.577
BE (0.157) with PSF	7.130	2.856	6.444	2.566
Lasso	7.159	2.893	6.477	2.589
Boosting	7.154	2.900	6.486	2.600

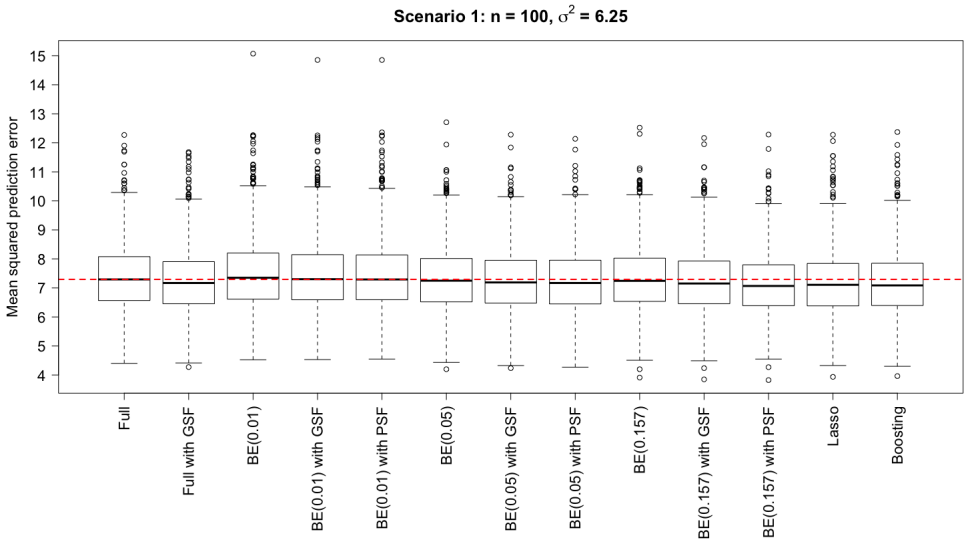

Figure 3.2: Boxplots of mean squared prediction error (MSPE) of different methods on a set of test data under scenario 1 ($n = 100, \sigma^2 = 6.25$).

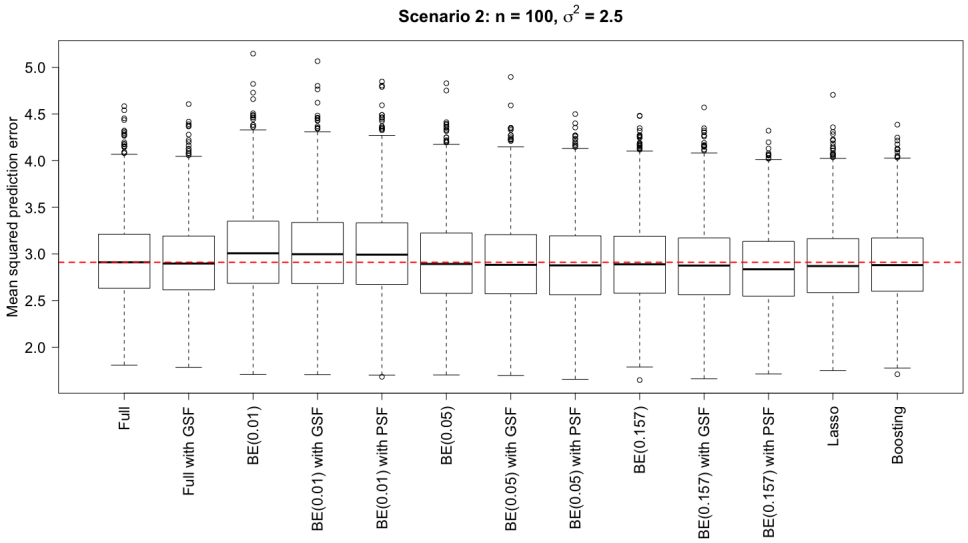

Figure 3.3: Boxplots of mean squared prediction error (MSPE) of different methods on a set of test data under scenario 2 ($n = 100, \sigma^2 = 2.5$).

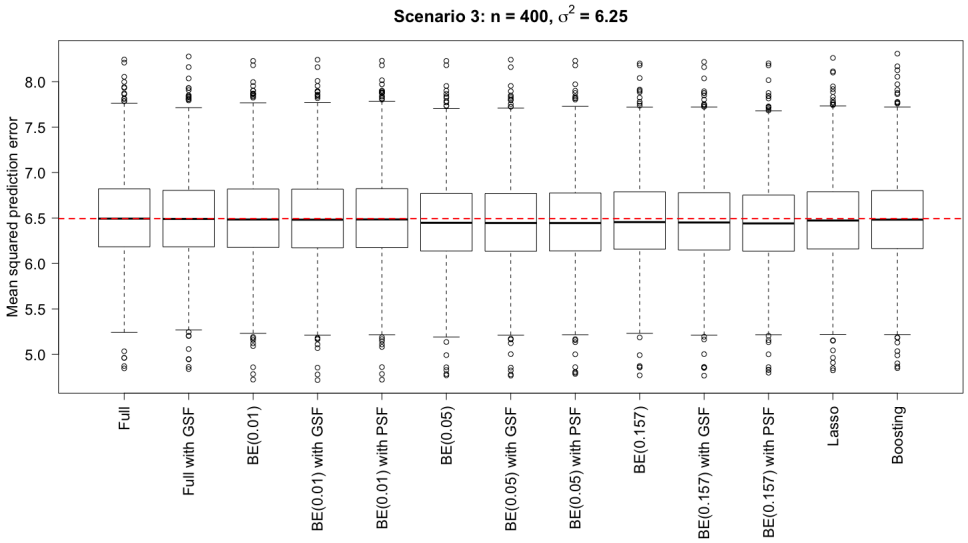

Figure 3.4: Boxplots of mean squared prediction error (MSPE) of different methods on a set of test data under scenario 3 ($n = 400, \sigma^2 = 6.25$).

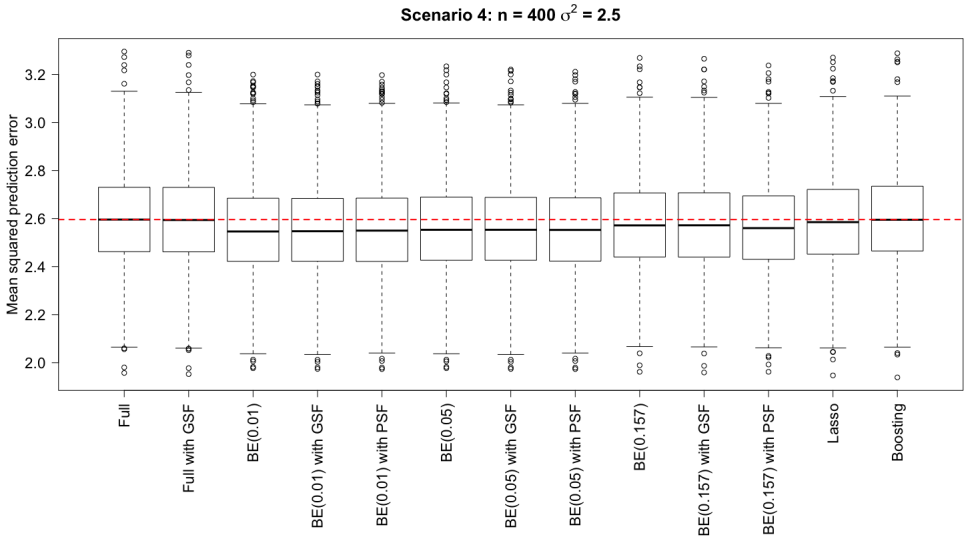

Figure 3.5: Boxplots of mean squared prediction error (MSPE) of different methods on a set of test data under scenario 4 ($n = 400, \sigma^2 = 2.5$).

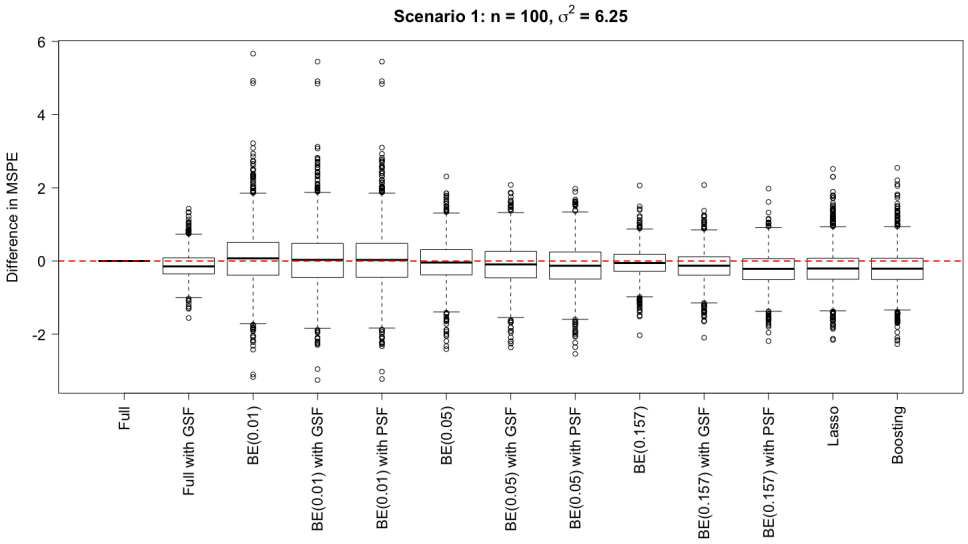

Figure 3.6: Boxplots of the differences in mean squared prediction error (MSPE) of different methods and least squares on a set of test data under scenario 1 ($n = 100, \sigma^2 = 6.25$).

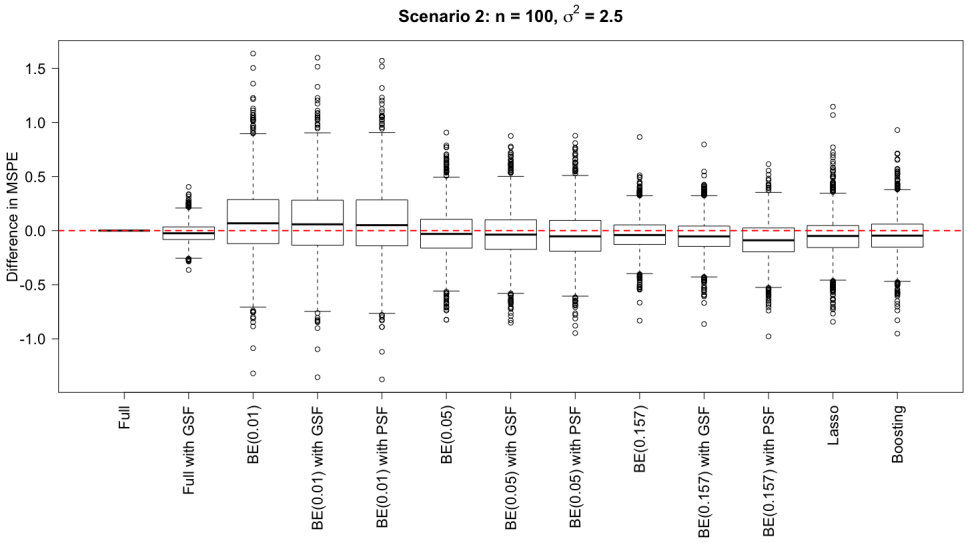

Figure 3.7: Boxplots of the differences in mean squared prediction error (MSPE) of different methods and least squares on a set of test data under scenario 2 ($n = 100, \sigma^2 = 2.5$).

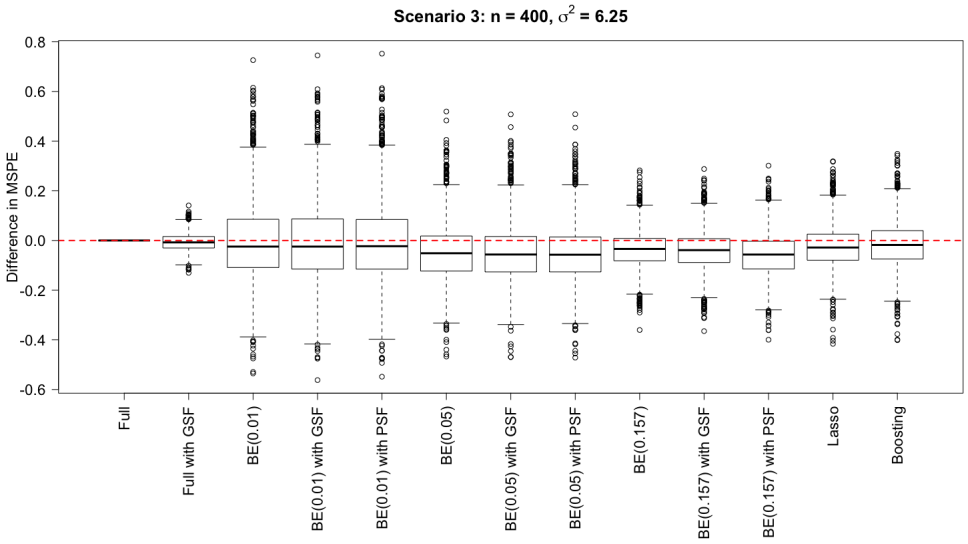

Figure 3.8: Boxplots of the differences in mean squared prediction error (MSPE) of different methods and least squares on a set of test data under scenario 3 ($n = 400, \sigma^2 = 6.25$).

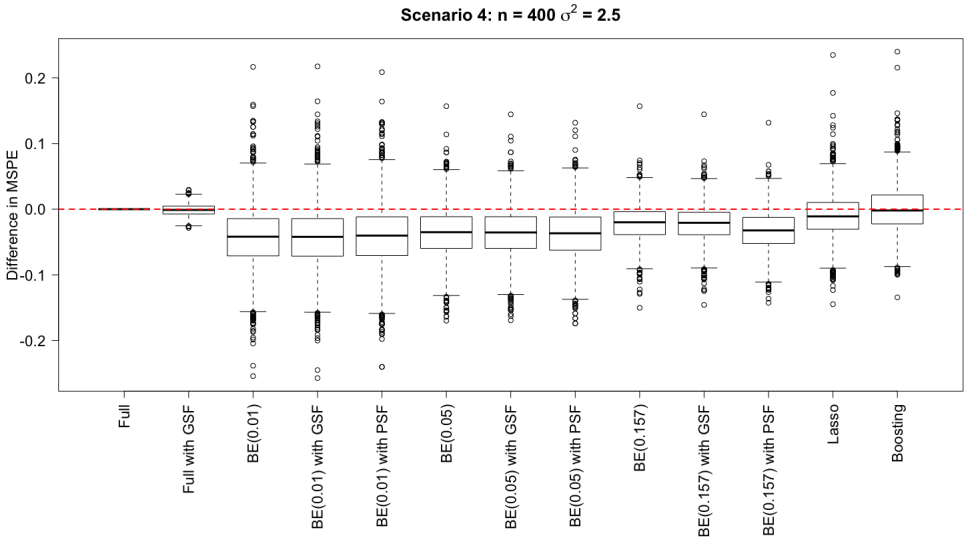

Figure 3.9: Boxplots of the differences in mean squared prediction error (MSPE) of different methods and least squares on a set of test data under scenario 4 ($n = 400, \sigma^2 = 2.5$).

3.3.2 Variable selection

Figure 3.10 shows the number of inclusion frequency of the covariates selected by different methods (i.e., BE with $\alpha = 0.01, 0.05, 0.157$, Lasso, L_2 boosting) under different scenarios. When the sample size is relatively small and variability is large (i.e., scenario 1), Lasso and boosting tend to produce models with a large number of covariates, including both relevant and redundant variables. On the contrary, BE tends to produce sparser models and the inclusion of the redundant variables is close to the nominal value (i.e., Type I error, here α).

When the sample size increases and the variability decreases (i.e., scenario 4), BE with three significance levels outperforms Lasso and boosting in terms of variable selection. Backward elimination produces models with almost all relevant variables. Lasso and boosting select all relevant variables, however, many redundant variables as well. In several repetitions, boosting selects the redundant variable x_1 which is correlated with the relevant variables x_5 and x_{10} with value 0.7 and 0.5, respectively. It may be concluded that boosting is not able to exclude from the final model redundant variables which are highly correlated to the relevant variables.

The present results also show that the BE method is sensitive to the sample size and variability. When comparing scenario 1 and 4, BE includes all relevant variables and reduces the frequency of including redundant variables. In contrast, the size and variability of the sample seems to have less impact on variable selection.

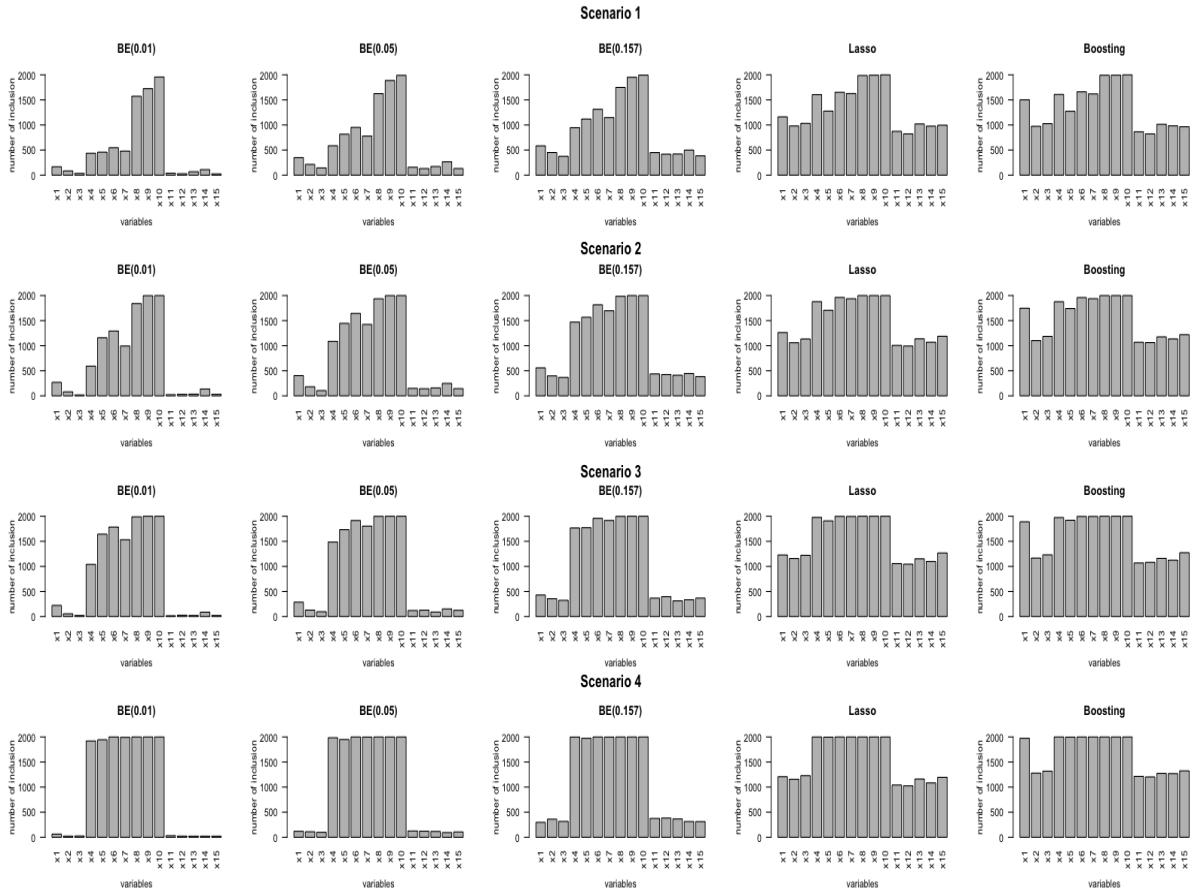

Figure 3.10: The number of inclusion for each candidate variable in 2,000 repetitions. The x-axis is the variables $x_1 - x_{15}$ and the y-axis is the number of inclusion of variables in 2,000 repetitions.

Earlier stop in boosting

Figure 3.11 illustrates the inclusion of variables in the models fitted by boosting with a smaller m_{stop}. The maximal number of iterations is set to 100 which is the default setting in the R-package mboost. Under each scenario, boosting tends to select only relevant variables and fewer redundant variables, whereas x_1 which is correlated to two relevant variables x_5 and x_{10} is included in several repetitions. In general, boosting identifies the relevant variables in the first iteration, while also tends to include redundant predictors in later steps. If the goal is to select correct variables for the model, an alternative way to select the best stopping criterion (m_{stop}) must be used instead of cross-validation.

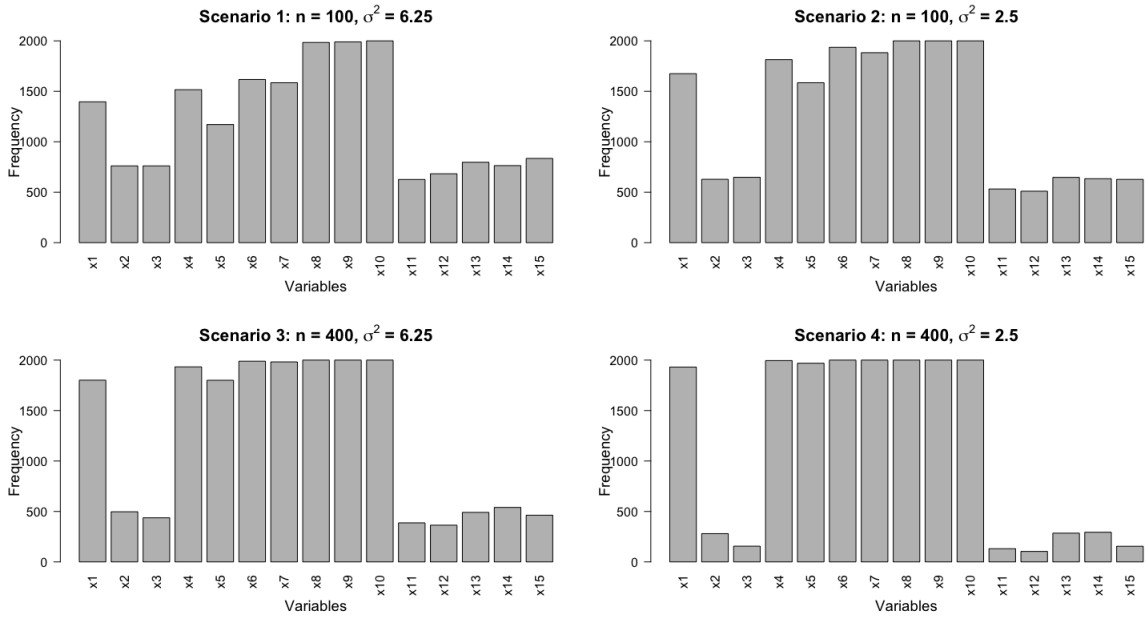

Figure 3.11: The number of inclusions for the variables in the model fitted by boosting with maximum iterations of 100.

Chapter 4

Artificial data example

In this chapter, the effects of independent variables are modelled by the FP functions. The corresponding model is fitted by the methods described in Chapter 2 on a set of simulated data [15]. The artificial (ART) dataset is simulated according to the characteristics of a real dataset from a breast cancer study [18]. This dataset is used as it contains the following characteristics which are useful for the present study: (i) the covariates have nonlinear effects; (ii) there is a sufficient number of observations; (iii) the outcome has a Gaussian distribution. The MSPE is used to evaluate the performances of different methods as described in Chapter 3.

4.1 Artificial data

The ART dataset contains 5,000 observations. Each observation has a continuous response variable y and 10 covariates $x_1, \ldots, x_{10}$ which consist of continuous $(x_1, x_3, x_5, x_6, x_7, x_{10})$, binary ($x_2$ and x_8) and categorical variables (x_4 and x_9). To avoid zero, the value of one is added to the predictor x_6 and x_7 for later FP analysis. Variables x_4 and x_9 have three levels (category 1, 2, 3), with x_4 being ordinal and x_9 being unordered (e.g., the risk of an individual to default a mortgage which can be categorized by high, moderate and low is considered as an ordinal variable, whereas the gender of an individual is considered as an unordered variable). Variable x_4 are coded into two dummy variables, x_{4a} (category 1 and 2) and x_{4b} (category 3). For x_9, category 2 and 3 are combined when coding into dummy variables, as category 3 is sparse.

Table 4.1 shows the distribution of continuous variables in the ART data. The response y, covariates x_1 and x_{10} are approximately normally distributed as the values of kurtosis and skewness are close to 3 which is the kurtosis of a normal distribution. The kurtosis and skewness of variables x_3, x_5, x_6 and x_7 are large, indicating that they have nonnormal distribution. The maximal values of x_5, x_6 and x_7 are much larger than the corresponding mean, indicating possible presence of influential data points.

Table 4.1: Distribution of continuous variables in the artificial (ART) dataset

Variable	Mean	SD	Min.	Max.	Skewness	Kurtosis	Missing
y	12.1	1.0	7.3	15.5	-0.1	3.1	0
x_1	54.5	10.0	20.0	88.0	0.0	3.0	0
x_3	21.4	9.1	5.0	82.0	1.2	5.5	0
x_5	6.8	32.4	0.2	1370.9	32.3	1221.7	0
x_6	146.7	221.1	1.0	3397.0	5.1	46.8	0
x_7	112.2	172.8	1.0	2441.0	5.0	43.5	0
x_{10}	16.8	8.2	0.1	64.9	0.8	4.1	0

4.2 Model fitting

The ART data is by random repeatedly divided to two sets, a training dataset containing 500 observations (a typical sample size as suggested by Royston and Sauerbrei [15]) and a test dataset containing 4,500 observations (the remaining observations from the dataset). The training data is used to fit the model with different methods. The test data is used to compute the MSPEs of the fitted models for evaluating the model performance.

In the fitting process, the MFP algorithm with the significance levels $\alpha_1 = 0.05$ and $\alpha_2 = 0.05$ is run for selecting the variables and choosing the FP transformation for continuous variables x_1, x_3, x_5, x_6, x_7 and x_{10}. The resulting model fitted by MFP is further modified with GSF, PSF and JSF, respectively. The L_2 boosting algorithm with FP base-learners (see Section 2.7) is applied to fit the training data. As the influential data points in x_5 have dominating effects on the resulting model, the base-learner for variable x_5 is fixed as $x_5^{-0.5}$ which is suggested by the robust analysis performed by Royston and Sauerbrei [15]. Two novel approaches which combine the MFP algorithm for the selection of FP functions with Lasso (MFP-Lasso) and with boosting (MFP-boosting) are also used to fit the model on the training data. The MSPEs of the fitted models are computed based on the test data. The fitting process is repeated 1,000 times. In each repetition, the training data with 500 observations is randomly chosen from the ART data. The true model for simulating the data is

$$f(x) = \beta_0 + \beta_{11} x_1^{0.5} + \beta_{12} x_1 + \beta_3 x_3 + \beta_{4a} x_{4a} + \beta_5 x_5^{-0.2} + \beta_6 log(x_6) + \beta_8 x_8 + \beta_{10} x_{10}.$$

4.3 Results and discussions

Figure 4.1 illustrates the distribution of the MSPEs of the fitted models over 1,000 repetitions. The MSPEs of the models fitted by a series of methods (MFP, MFP-shrinkage factors, MFP-Lasso, MFP-boosting) are close to zero, indicating reasonable prediction performances. Figure 4.2 shows that the differences in MSPE among the methods are quite small, except for boosting with FP base-learners, as most of the medians are close to zero. The effects of GSF and PSF on the models fitted by MFP are fairly similar. Table 4.2 shows the average

and standard deviation of the MSPE of 1,000 repetitions. The mean of the model with JSF is slightly smaller than that of the MFP model. The results reveal the effects of JSF on improving the prediction accuracy of the MFP model. The fitted models produced by MFP-Lasso and MFP-boosting tend to have MSPEs similar to MFP.

A relevant number of values for the MSPE of the models fitted by boosting with FP base-learners (right plot of Figure 4.1) are large, resulting in a large average and large standard deviation in Table 4.2. The prediction performance of this implementation of L_2 boosting is worse and unstable compared to other models. When evaluating the prediction error on the training data, boosting with the FP base-learners has the smallest averaged mean squared residuals over 1,000 repetitions (data not shown), which may indicate overfitting. However, this is probably not the most serious problem of this method. As the number of base-learners for each variable is large (16 base-learners), boosting chooses a lot of different functional forms. As a result, at least one function is often incorrectly chosen, which tends to lead to a bad prediction performance of this method. The highest value of MSPE obtained by the other methods (not only those displayed in the left plot in Figure 4.1, but especially some more excluded from the plot for displaying reason) corresponds to iterations in which MFP selects the wrong FP, meaning that selecting the wrong FP leads to a bad prediction performance.

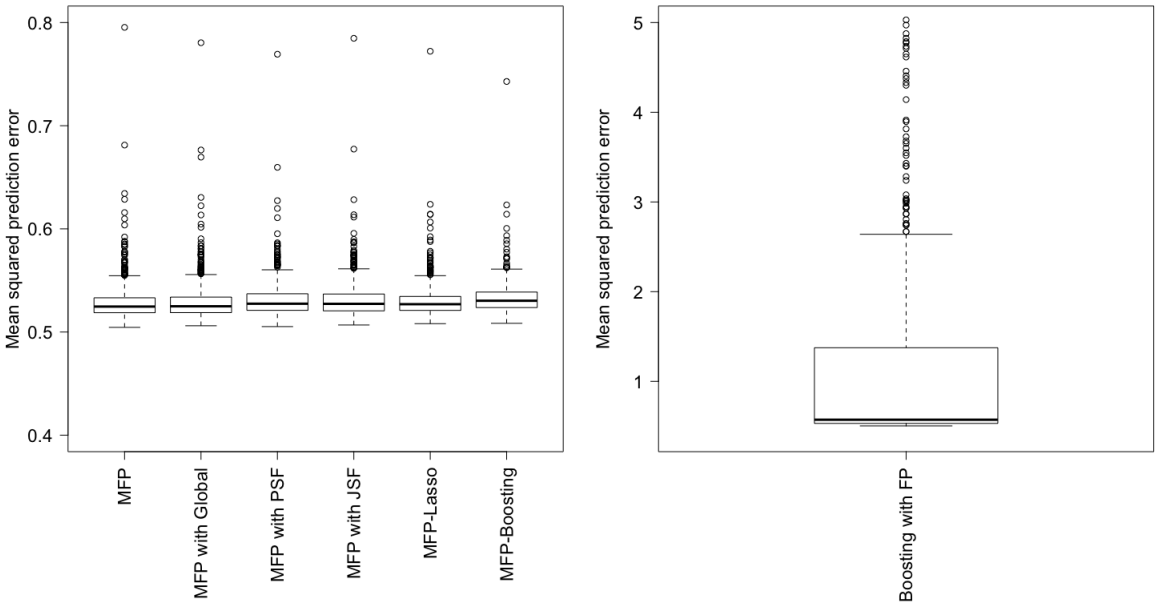

Figure 4.1: Boxplots displaying the mean squared prediction errors (MSPEs) of fitted models on the test data. The few largest values on the figures are not included.

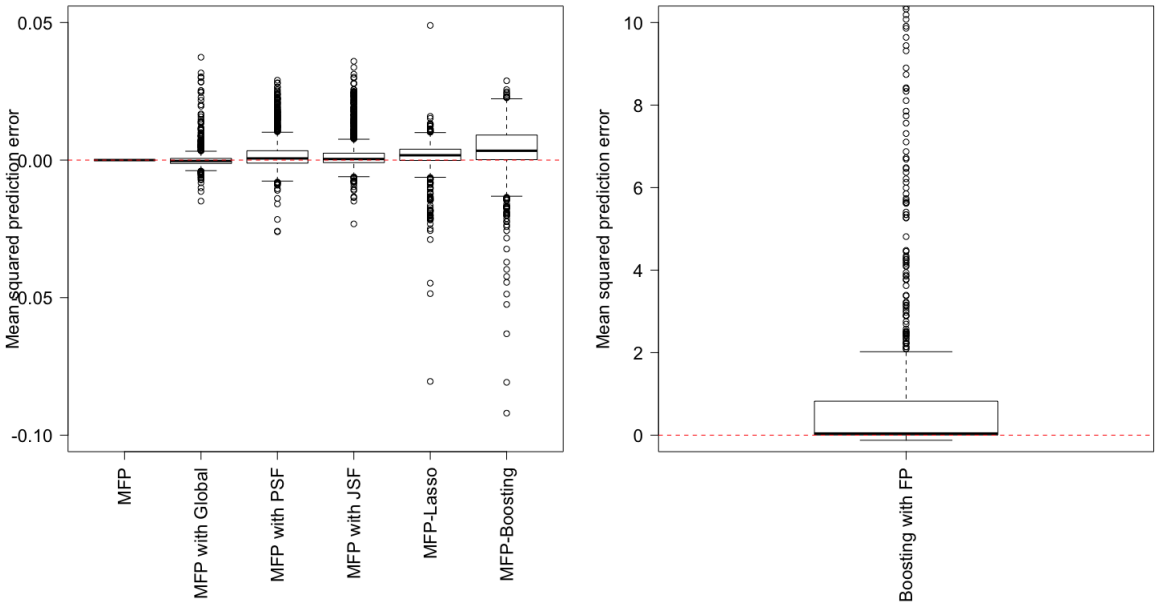

Figure 4.2: Boxplots displaying the differences in mean squared prediction errors (MSPEs) between fitted models and multivariable model-building with fractional polynomials (MFP).

Table 4.2: Mean squared prediction errors (MSPEs) and standard deviations of the fitted models over 1,000 repetitions. MFP: multivariable model-building with fractional polynomials; GSF: global shrinkage factors; PSF: parameterwise shrinkage factors; JSF: joint shrinkage factors; FP: fractional polynomials; MSPE: mean squared prediction error.

Method	MSPE	Standard deviation
MFP	0.542	0.312
MFP with GSF	0.539	0.233
MFP with PSF	0.547	0.361
MFP with JSF	0.541	0.229
Boosting with FP base-learners	12.952	105.464
MFP-Lasso	0.543	0.308
MFP-Boosting	0.542	0.214

Chapter 5

Conclusions

In this thesis, the prediction performances of a series of methods, namely shrinkage factors (global, parameterwise and joint shrinkage), Lasso and boosting (with and without FP) are evaluated and compared. The performances of these methods are contrasted in two case studies, in which different types of effects, linear and a combination of linear and nonlinear effects are present. The MSPE computed on an independent test set is used as an evaluation criterion.

In the case of only linear effects, the results show that post-selection shrinkage factors may improve the prediction performance of the least squares method when the data contains a lot of information and does not have much unexplainable variability. When the sample size is small and the variability is large, boosting and Lasso can improve the prediction performance of the least squares method. The significance level has an impact on the prediction performance of the post-selection shrinkage methods. All in all, the present study suggests that a combination of BE with the significance level $\alpha = 0.157$ (AIC) and PSF may in general yield the best prediction result.

In the case of a combination of linear and nonlinear effects of independent variables, the results show that L_2 with FP base-learners has a bad performance in terms of prediction. For each variable, it selects a number of functions which may lead to poor prediction accuracy. For building an FP model, L_2 boosting with the FP base-learners implemented in the R-package mboost is thus not recommended. When contacting the developer of the method (T. Hothorn) for an opinion, he confirmed that using FP within the boosting algorithm is "a bad idea". Other methods, such as MFP with shrinkage factors, MFP-boosting and MFP-Lasso, have similar prediction performances as the MFP algorithm, but show some potential.

The latter two methods, in particular, are novel and it may be worth performing further investigations in the future. For example, a better way to select the functional forms without the need to implement the MFP algorithm, may be developed. Future extensions of the present study may consider other approaches, such as boosting with splines as the base-learners which is currently widely used in practice in contrast to the less-studied boosting with FP approach.

Bibliography

[1] L. Breiman. Arcing classifier. *The Annals of Statistics*, 26:801–849, 1998.

[2] L. Breiman. Prediction games and arcing algorithms. *Neural Computation*, 11:1493–1517, 1999.

[3] P. Bühlmann and T. Hothorn. Boosting algorithms: regularization, prediction and model fitting. *Statistical Science*, 22:477–505, 2007.

[4] P. Bühlmann and B. Yu. Boosting with the l_2 loss. *Journal of the American Statistical Association*, 98:324–339, 2003.

[5] J. B. Copas. Regression, prediction and shrinkage. *Journal of the Royal Statistical Society. Series B (Methodological)*, 45:311–354, 1983.

[6] D. Dunkler, W. Sauerbrei, and G. Heinze. Global, parameterwise and joint shrinkage factor estimation. *Journal of Statistical Software*, 69:1–19, 2016.

[7] J. Friedman. Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, 29:1189–1232, 2001.

[8] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 28:337–407, 2000.

[9] J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33:1–22, 2010.

[10] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning*. Springer, 2009.

[11] B. Hofner, A. Mayr, N. Robinzonov, and M. Schmid. Model-based boosting in r: a hands-on tutorial using the r package mboost. *Computational Statistics*, 29:3–35, 2014.

[12] H. V. Houwelingen and W. Sauerbrei. Cross-validation, shrinkage and variable selection in linear regression revisited. *Open Journal of Statistics*, 3:79–102, 2013.

[13] J. C. V. Houwelingen and S. L. Cessie. Predictive value of statistical models. *Statistics in Medicine*, 9:1303–1325, 1990.

[14] P. Royston and D. G. Altman. Regression using fractional polynomials of continuous covariates: parsimonious parametric modelling. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 43:429–467, 1994.

[15] P. Royston and W. Sauerbrei. *Multivariable model-building: a pragmatic approach to regression anaylsis based on fractional polynomials for modelling continuous variables*. John Wiley & Sons, 2008.

[16] W. Sauerbrei. The use of resampling methods to simplify regression models in medical statistics. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 48:313–329, 1999.

[17] W. Sauerbrei and P. Royston. Building multivariable prognostic and diagnostic models: transformation of the predictors by using fractional polynomials. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 162:71–94, 1999.

[18] M. Schumacher, G. Basert, H. Bojar, K. Huebner, M. Olschewski, W. Sauerbrei, C. Schmoor, C. Beyerle, R. Neumann, and H. Rauschecker. Randomized 2×2 trial evaluating hormonal treatment and the duration of chemotherapy in node-positive breast cancer patients. *Journal of Clinical Oncology*, 12:2086–2093, 1994.

Appendix A

R scripts

All the R codes used for the case studies in this thesis are listed as the following.

A.1 R codes for the small simulation study

```
# Import necessary packages
library(MASS)
library(glmnet)
library(shrink)
library(mboost)
# Define the necessary variables
N = 2000
n = c(100, 400)
sigma = c(sqrt(6.25), sqrt(2.5))
mu = rep(0, 15)
true.beta = 0*seq(1, 15)
true.beta[4] = -0.5
true.beta[5:7] = 0.5
true.beta[8:9] = 1
true.beta[10] = 1.5
Sigma = diag(x = 1, nrow = 15, ncol = 15)
Sigma[1,5] = Sigma[5,1] = Sigma[11,12] = Sigma[12,11] = 0.7
Sigma[7,14] = Sigma[14,7] = Sigma[1,10] = Sigma[10,1] =
Sigma[2,6] = Sigma[6,2] = Sigma[9, 13] = Sigma[13,9] = 0.5
Sigma[4,8] = Sigma[8,4] = -0.7
Sigma[7,8] = Sigma[8,7] = 0.3
mspe = matrix(0,nrow = N, ncol = 52)
mse = matrix(0,nrow = N, ncol = 52)
varinclusion.BE = matrix(0,nrow = 15,ncol = 12)
varinclusion.boosting = matrix(0,nrow = 15,ncol = 4)
varinclusion.lasso = matrix(0,nrow = 15,ncol = 4)
alpha = c(0.01,0.05,0.157)

# Fitting process
for (i in 1:2){
```

```
  print(paste0("n = ",n[i]))
  for (j in 1:2){
    print(paste0("sigma = ",sigma[j]))
    for (k in 1:N){
      x.train = mvrnorm(n[i],mu,Sigma)
      y.train = x.train%*%true.beta+rnorm(n[i],0,sigma[j])
      x.test = mvrnorm(n[i],mu,Sigma)
      y.test = x.test%*%true.beta+rnorm(n[i],0,sigma[j])
      ## Full model via OLS
      lr.full = lm(y.train~.-1, data = as.data.frame(x.train),x = TRUE,y =
          TRUE)
      mse[k,(i-1)*26+(j-1)*13+1] = mean(lr.full$residuals^2)
      mspe[k,(i-1)*26+(j-1)*13+1] = mean((y.test-predict(lr.full,newdata = as.
          data.frame(x.test),type = "response"))^2)
      ## Full model with a global shrinkage factor
      full.global = shrink(lr.full, type = "global", method = "jackknife")
      mse[k,(i-1)*26+(j-1)*13+2] = mean((y.train-predict(full.global,newdata =
           as.data.frame(x.train),type = "response"))^2)
      mspe[k,(i-1)*26+(j-1)*13+2] = mean((y.test-predict(full.global,newdata =
           as.data.frame(x.test),type = "response"))^2)
      ## BE with shrinkage
      for (h in 1:3){
        lr.variable.selection = as.data.frame(drop1(lr.full, test = "Chisq"))
            [-1,]
        new.model = lr.full
        while(max(lr.variable.selection[, "Pr(>Chi)"])>alpha[h]){
          most.insig.row = which.max(lr.variable.selection[, "Pr(>Chi)"])
          most.insig.var = rownames(lr.variable.selection)[most.insig.row]
          old.formula = formula(new.model)
          new.formula = update(old.formula,paste(".~. -", most.insig.var))
          new.model = lm(formula = new.formula,data = as.data.frame(x.train),
              x = TRUE, y = TRUE)
          lr.variable.selection = as.data.frame(drop1(new.model, test = "Chisq
              "))[-1,]
        }
        for (m in as.numeric(gsub(".*?([0-9]+).*$","\\1",rownames(as.data.
            frame(new.model$coefficients))))){
          varinclusion.BE[m,(i-1)*6+(j-1)*3+h] = varinclusion.BE[m,(i-1)*6+(j
              -1)*3+h]+1
        }
        ## BE
        mse[k,(i-1)*26+(j-1)*13+3+(h-1)*3] = mean((y.train-predict(new.model,
            newdata = as.data.frame(x.train),type = "response"))^2)
        mspe[k,(i-1)*26+(j-1)*13+3+(h-1)*3] = mean((y.test-predict(new.model,
            newdata = as.data.frame(x.test),type = "response"))^2)
        ## BE with GSF
        BE.global = shrink(new.model, type = "global", method = "jackknife")
        mse[k,(i-1)*26+(j-1)*13+4+(h-1)*3] = mean((y.train-predict(BE.global,
            newdata = as.data.frame(x.train),type = "response"))^2)
        mspe[k,(i-1)*26+(j-1)*13+4+(h-1)*3] = mean((y.test-predict(BE.global,
            newdata = as.data.frame(x.test),type = "response"))^2)
```

```
69          ## BE with PSF
70          BE.pw = shrink(new.model, type = "parameterwise", method = "jackknife"
                )
71          mse[k,(i-1)*26+(j-1)*13+5+(h-1)*3] = mean((y.train-predict(BE.pw,
                newdata = as.data.frame(x.train),type = "response"))^2)
72          mspe[k,(i-1)*26+(j-1)*13+5+(h-1)*3] = mean((y.test-predict(BE.pw,
                newdata = as.data.frame(x.test),type = "response"))^2)
73        }
74        ## Lasso
75        cv.fit = cv.glmnet(x = x.train, y = y.train, nfolds = 10, alpha = 1)
76        lambda.hat = cv.fit$lambda.min
77        lasso.fit = glmnet(x = x.train, y = y.train, alpha = 1, lambda = lambda.
              hat, family = "gaussian")
78        for (m in which(lasso.fit$beta! = 0)){
79          varinclusion.lasso[m,(i-1)*2+j] = varinclusion.lasso[m,(i-1)*2+j]+1
80        }
81        mse[k,(i-1)*26+(j-1)*13+12] = mean((y.train-predict(lasso.fit,newx = x.
              train,s = lambda.hat,type = "response"))^2)
82        mspe[k,(i-1)*26+(j-1)*13+12] = mean((y.test-predict(lasso.fit,newx = x.
              test,s = lambda.hat,type = "response"))^2)
83        ## Boosting
84        boosting.fit = glmboost(y = as.vector(y.train), x = x.train, family =
              Gaussian(), control = boost_control(mstop = 500))
85        cvm=cvrisk(boosting.fit, folds = cv(model.weights(boosting.fit), type =
              "kfold", B = 10))
86        boosting.fit[mstop(cvm)]
87        for (m in as.numeric(gsub(".*?([0-9]+).*$","\\1",names(coef(boosting.fit
              [mstop(cvm)])))))){
88          varinclusion.boosting[m,(i-1)*2+j] = varinclusion.boosting[m,(i-1)*2+j
                ]+1
89        }
90        mse[k,(i-1)*26+(j-1)*13+13] = mean(boosting.fit[mstop(cvm)]$resid()^2)
91        mspe[k,(i-1)*26+(j-1)*13+13] = mean((y.test-predict(boosting.fit[mstop(
              cvm)],newdata = x.test,type = "response"))^2)
92      }
93    }
94  }
95  # Average of the mean squared prediction error
96  avg.mse = apply(mse,2,mean)
97  avg.mspe = apply(mspe,2,mean)
98
99  # Boosting with the maximal iterations in 100
100 varinclusion.boosting.early = matrix(0,nrow = 15,ncol = 4)
101 mspe.early = matrix(0,nrow = N,ncol = 4)
102 istop = matrix(0,nrow = N,ncol = 4)
103 for (i in 1:2){
104   print(paste0("n = ",n[i]))
105   for (j in 1:2){
106     print(paste0("sigma = ",sigma[j]))
107     for (k in 1:N){
108       x.train = mvrnorm(n[i],mu,Sigma)
```

```
      y.train = x.train%*%true.beta+rnorm(n[i],0,sigma[j])
      x.test = mvrnorm(n[i],mu,Sigma)
      y.test = x.test%*%true.beta+rnorm(n[i],0,sigma[j])
      boosting.early.fit = glmboost(y = as.vector(y.train), x = x.train,
          family = Gaussian())
      cvm = cvrisk(boosting.early.fit, folds = cv(model.weights(boosting.early
          .fit), type = "kfold", B = 10))
      boosting.early.fit[mstop(cvm)]
      istop[k,(i-1)*2+j] = mstop(cvm)
      for (m in as.numeric(gsub(".*?([0-9]+).*$","\\1",names(coef(boosting.
          early.fit[mstop(cvm)])))){
        varinclusion.boosting.early[m,(i-1)*2+j] = varinclusion.boosting.early
            [m,(i-1)*2+j]+1
      }
      mspe.early[k,(i-1)*2+j] = mean((y.test-predict(boosting.early.fit[mstop(
          cvm)],newdata = x.test,type = "response"))^2)
    }
  }
}

# Boxplots of MSPE
par(oma = c(6.5,0,0,0),las = 2,cex.axis = 1.4,cex.lab = 1.4,cex.main = 1.6)
# Scenario 1
boxplot(mspe[,1:13],xaxt = "n",yaxt = "n")
abline(h = median(mspe[,1]),lty = 2,lwd = 2,col = "red")
axis(2,at = seq(4.0,15.0,by = 1))
axis(1,at = 1:13,
     labels = c("Full","Full with GSF",
                "BE(0.01)","BE(0.01) with GSF","BE(0.01) with PSF",
                "BE(0.05)","BE(0.05) with GSF","BE(0.05) with PSF",
                "BE(0.157)","BE(0.157) with GSF","BE(0.157) with PSF",
                "Lasso","Boosting"))
title(main = expression(bold(paste("Scenario 1: n = 100, ", sigma^2, " = 6.25"
    ))),
      ylab = "Mean squared prediction error")
# Scenario 2
boxplot(mspe[,14:26],xaxt = "n")
abline(h = median(mspe[,14]),lty = 2,lwd = 2,col = "red")
axis(1,at = 1:13,
     labels = c("Full","Full with GSF",
                "BE(0.01)","BE(0.01) with GSF","BE(0.01) with PSF",
                "BE(0.05)","BE(0.05) with GSF","BE(0.05) with PSF",
                "BE(0.157)","BE(0.157) with GSF","BE(0.157) with PSF",
                "Lasso","Boosting"))
title(main = expression(bold(paste("Scenario 2: n = 100, ", sigma^2, " = 2.5")
    )),
      ylab = "Mean squared prediction error")
# Scenario 3
boxplot(mspe[,27:39],xaxt = "n")
abline(h = median(mspe[,27]),lty = 2,lwd = 2,col = "red")
axis(1,at = 1:13,
```

```
153      labels = c("Full","Full with GSF",
154                 "BE(0.01)","BE(0.01) with GSF","BE(0.01) with PSF",
155                 "BE(0.05)","BE(0.05) with GSF","BE(0.05) with PSF",
156                 "BE(0.157)","BE(0.157) with GSF","BE(0.157) with PSF",
157                 "Lasso","Boosting"))
158 title(main = expression(bold(paste("Scenario 3: n = 400, ", sigma^2, " = 6.25"
    ))),
159       ylab = "Mean squared prediction error")
160 # Scenario 4
161 boxplot(mspe[,40:52],xaxt = "n")
162 abline(h = median(mspe[,40]),lty = 2,lwd = 2,col = "red")
163 axis(1,at = 1:13,
164      labels = c("Full","Full with GSF",
165                 "BE(0.01)","BE(0.01) with GSF","BE(0.01) with PSF",
166                 "BE(0.05)","BE(0.05) with GSF","BE(0.05) with PSF",
167                 "BE(0.157)","BE(0.157) with GSF","BE(0.157) with PSF",
168                 "Lasso","Boosting"))
169 title(main = expression(bold(paste("Scenario 4: n = 400 ", sigma^2, " = 2.5"))
    ),
170       ylab = "Mean squared prediction error")
171
172 # Boxplots of the differences in MSPE between all methods and full
173 par(oma = c(6.5,0,0,0),las = 2,cex.axis = 1.4,cex.lab = 1.4,cex.main = 1.6)
174 # Scenario 1
175 boxplot(mspe[,1:13]-mspe[,1],xaxt = "n")
176 abline(h = 0,lty = 2,col = "red",lwd = 2)
177 axis(1,at = 1:13,
178      labels = c("Full","Full with GSF",
179                 "BE(0.01)","BE(0.01) with GSF","BE(0.01) with PSF",
180                 "BE(0.05)","BE(0.05) with GSF","BE(0.05) with PSF",
181                 "BE(0.157)","BE(0.157) with GSF","BE(0.157) with PSF",
182                 "Lasso","Boosting"))
183 title(main = expression(bold(paste("Scenario 1: n = 100, ", sigma^2, " = 6.25"
    ))),
184       ylab = "Difference in MSPE")
185 # Scenario 2
186 boxplot(mspe[,14:26]-mspe[,14],xaxt = "n")
187 abline(h = 0,lty = 2,col = "red",lwd = 2)
188 axis(1,at = 1:13,
189      labels = c("Full","Full with GSF",
190                 "BE(0.01)","BE(0.01) with GSF","BE(0.01) with PSF",
191                 "BE(0.05)","BE(0.05) with GSF","BE(0.05) with PSF",
192                 "BE(0.157)","BE(0.157) with GSF","BE(0.157) with PSF",
193                 "Lasso","Boosting"))
194 title(main = expression(bold(paste("Scenario 2: n = 100, ", sigma^2, " = 2.5")
    )),
195       ylab = "Difference in MSPE")
196 # Scenario 3
197 boxplot(mspe[,27:39]-mspe[,27],xaxt = "n")
198 abline(h = 0,lty = 2,col = "red",lwd = 2)
199 axis(1,at = 1:13,
```

```
     labels = c("Full","Full with GSF",
                "BE(0.01)","BE(0.01) with GSF","BE(0.01) with PSF",
                "BE(0.05)","BE(0.05) with GSF","BE(0.05) with PSF",
                "BE(0.157)","BE(0.157) with GSF","BE(0.157) with PSF",
                "Lasso","Boosting"))
title(main = expression(bold(paste("Scenario 3: n = 400, ", sigma^2, " = 6.25"
    ))),
      ylab = "Difference in MSPE")
# Scenaro 4
boxplot(mspe[,40:52]-mspe[,40],xaxt = "n")
abline(h = 0,lty = 2,col = "red",lwd = 2)
axis(1,at = 1:13,
     labels = c("Full","Full with GSF",
                "BE(0.01)","BE(0.01) with GSF","BE(0.01) with PSF",
                "BE(0.05)","BE(0.05) with GSF","BE(0.05) with PSF",
                "BE(0.157)","BE(0.157) with GSF","BE(0.157) with PSF",
                "Lasso","Boosting"))
title(main = expression(bold(paste("Scenario 4: n = 400 ", sigma^2, " = 2.5"))
    ),
      ylab = "Difference in MSPE")

# Inclusion frequency
par(mfrow = c(4,5),oma = c(0,0,2,0),las = 2)
# Scenario 1
barplot(varinclusion.BE[,1],ylim = c(0,2000),
        names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                      "x9","x10","x11","x12","x13","x14","x15"),
        xlab = "Variables",ylab = "Frequency",main = "BE(0.01)")
mtext("Scenario 1",side = 3,las = 0,outer = TRUE,font = 2)
barplot(varinclusion.BE[,2]
        ,ylim = c(0,2000)
        ,names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                       "x9","x10","x11","x12","x13","x14","x15")
        ,xlab = "Variables",ylab = "Frequency"
        ,main = "BE(0.05)")
barplot(varinclusion.BE[,3]
        ,ylim = c(0,2000)
        ,names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                       "x9","x10","x11","x12","x13","x14","x15")
        ,xlab = "Variables",ylab = "Frequency"
        ,main = "BE(0.157)")
barplot(varinclusion.lasso[,1]
        ,ylim = c(0,2000)
        ,names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                       "x9","x10","x11","x12","x13","x14","x15")
        ,xlab = "Variables",ylab = "Frequency"
        ,main = "Lasso")
barplot(varinclusion.boosting[,1]
        ,ylim = c(0,2000)
        ,names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                       "x9","x10","x11","x12","x13","x14","x15")
```

```
        ,xlab = "Variables",ylab = "Frequency"
        ,main = "Boosting")

# Scenario 2
barplot(varinclusion.BE[,4]
        ,ylim = c(0,2000)
        ,names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                       "x9","x10","x11","x12","x13","x14","x15")
        ,xlab = "Variables",ylab = "Frequency"
        ,main = "BE(0.01)")
mtext("Scenario 2",side = 3,line = -18,outer = TRUE,las = 0,font = 2)
barplot(varinclusion.BE[,5]
        ,ylim = c(0,2000)
        ,names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                       "x9","x10","x11","x12","x13","x14","x15")
        ,xlab = "Variables",ylab = "Frequency"
        ,main = "BE(0.05)")
barplot(varinclusion.BE[,6]
        ,ylim = c(0,2000)
        ,names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                       "x9","x10","x11","x12","x13","x14","x15")
        ,xlab = "Variables",ylab = "Frequency"
        ,main = "BE(0.157)")
barplot(varinclusion.lasso[,2]
        ,ylim = c(0,2000)
        ,names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                       "x9","x10","x11","x12","x13","x14","x15")
        ,xlab = "Variables",ylab = "Frequency"
        ,main = "Lasso")
barplot(varinclusion.boosting[,2]
        ,ylim = c(0,2000)
        ,names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                       "x9","x10","x11","x12","x13","x14","x15")
        ,xlab = "Variables",ylab = "Frequency"
        ,main = "Boosting")

# Scenario 3
barplot(varinclusion.BE[,7]
        ,ylim = c(0,2000)
        ,names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                       "x9","x10","x11","x12","x13","x14","x15")
        ,xlab = "Variables",ylab = "Frequency"
        ,main = "BE(0.01)")
mtext("Scenario 3",side = 3,line = -35,las = 0,outer = TRUE,font = 2)
barplot(varinclusion.BE[,8]
        ,ylim = c(0,2000)
        ,names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                       "x9","x10","x11","x12","x13","x14","x15")
        ,xlab = "Variables",ylab = "Frequency"
        ,main = "BE(0.05)")
barplot(varinclusion.BE[,9]
```

```
        ,ylim = c(0,2000)
        ,names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                       "x9","x10","x11","x12","x13","x14","x15")
        ,xlab = "Variables",ylab = "Frequency"
        ,main = "BE(0.157)")
barplot(varinclusion.lasso[,3]
        ,ylim = c(0,2000)
        ,names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                       "x9","x10","x11","x12","x13","x14","x15")
        ,xlab = "Variables",ylab = "Frequency"
        ,main = "Lasso")
barplot(varinclusion.boosting[,3]
        ,ylim = c(0,2000)
        ,names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                       "x9","x10","x11","x12","x13","x14","x15")
        ,xlab = "Variables",ylab = "Frequency"
        ,main = "Boosting")

# Scenario 4
barplot(varinclusion.BE[,10]
        ,ylim = c(0,2000)
        ,names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                       "x9","x10","x11","x12","x13","x14","x15")
        ,xlab = "Variables",ylab = "Frequency"
        ,main = "BE(0.01)")
mtext("Scenario 4",side = 3,line = -51,las = 0,outer = TRUE,font = 2)
barplot(varinclusion.BE[,11]
        ,ylim = c(0,2000)
        ,names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                       "x9","x10","x11","x12","x13","x14","x15")
        ,xlab = "Variables",ylab = "Frequency"
        ,main = "BE(0.05)")
barplot(varinclusion.BE[,12]
        ,ylim = c(0,2000)
        ,names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                       "x9","x10","x11","x12","x13","x14","x15")
        ,xlab = "Variables",ylab = "Frequency"
        ,main = "BE(0.157)")
barplot(varinclusion.lasso[,4]
        ,ylim = c(0,2000)
        ,names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                       "x9","x10","x11","x12","x13","x14","x15")
        ,xlab = "Variables",ylab = "Frequency"
        ,main = "Lasso")
barplot(varinclusion.boosting[,4]
        ,ylim = c(0,2000)
        ,names.arg = c("x1","x2","x3","x4","x5","x6","x7","x8",
                       "x9","x10","x11","x12","x13","x14","x15")
        ,xlab = "Variables",ylab = "Frequency"
        ,main = "Boosting")

```

```
# Inclusion frequency of boosting with 100 iterations
par(mfrow = c(2,2),las = 2,cex.lab = 1.4,cex.main = 1.6,cex.axis = 1.4)
# Scenario 1
barplot(varinclusion.boosting.early[,1],ylim = c(0,2000)
        ,names.arg = c(expression("x"[1]),expression("x"[2]),expression("x"
            [3]),
                       expression("x"[4]),expression("x"[5]),expression("x"
                           [6]),
                       expression("x"[7]),expression("x"[8]),expression("x"
                           [9]),
                       expression("x"[10]),expression("x"[11]),expression("x"
                           [12]),
                       expression("x"[13]),expression("x"[14]),expression("x"
                           [15])))
title(main = expression(bold(paste("Scenario 1: n = 100, ", sigma^2, " = 6.25"
    ))),
      xlab = "Variables",ylab = "Frequency")
# Scenario 2
barplot(varinclusion.boosting.early[,2],ylim = c(0,2000)
        ,names.arg = c(expression("x"[1]),expression("x"[2]),expression("x"
            [3]),
                       expression("x"[4]),expression("x"[5]),expression("x"
                           [6]),
                       expression("x"[7]),expression("x"[8]),expression("x"
                           [9]),
                       expression("x"[10]),expression("x"[11]),expression("x"
                           [12]),
                       expression("x"[13]),expression("x"[14]),expression("x"
                           [15])))
title(main = expression(bold(paste("Scenario 2: n = 100, ", sigma^2, " = 2.5")
    )),
      xlab = "Variables",ylab = "Frequency")
# Scenario 3
barplot(varinclusion.boosting.early[,3],ylim = c(0,2000)
        ,names.arg = c(expression("x"[1]),expression("x"[2]),expression("x"
            [3]),
                       expression("x"[4]),expression("x"[5]),expression("x"
                           [6]),
                       expression("x"[7]),expression("x"[8]),expression("x"
                           [9]),
                       expression("x"[10]),expression("x"[11]),expression("x"
                           [12]),
                       expression("x"[13]),expression("x"[14]),expression("x"
                           [15])))
title(main = expression(bold(paste("Scenario 3: n = 400, ", sigma^2, " = 6.25"
    ))),
      xlab = "Variables",ylab = "Frequency")
# Scenario 4
barplot(varinclusion.boosting.early[,4],ylim = c(0,2000)
        ,names.arg = c(expression("x"[1]),expression("x"[2]),expression("x"
            [3]),
```

```
                        expression("x"[4]),expression("x"[5]),expression("x"
                            [6]),
                        expression("x"[7]),expression("x"[8]),expression("x"
                            [9]),
                        expression("x"[10]),expression("x"[11]),expression("x"
                            [12]),
                        expression("x"[13]),expression("x"[14]),expression("x"
                            [15]))
        )
title(main = expression(bold(paste("Scenario 4: n = 400, ", sigma^2, " = 2.5")
    )),
      xlab = "Variables",ylab = "Frequency")
```

A.2 R codes for the artificial data example

```
# Import necessary packages
library(mfp)
library(shrink)
library(mboost)
library(glmnet)
# ART data analysis
art=read.csv(file="~ART/art1/art1.csv")
art.mod=art[c("x1","x2","x3","x4","x5","x6","x7","x8","x9","x10","x4a","x4b","
    x9a","x9b","y")]
art.mod$x6=art.mod$x6+1
art.mod$x7=art.mod$x7+1
m=1000
test.error=as.data.frame(matrix(0,nrow=m,ncol = 7))
sq.residual=as.data.frame(matrix(0,nrow=m,ncol=7))
colnames(test.error)=c("No","Global","Parameterwise","Join", "Boosting",
                       "Lasso","Modified boosting")
# Fitting process
for (j in 1:m){
  print(j)
  list_join=list()
  set.seed(j)
  train.index=sample(1:nrow(art.mod),size=500)
  train=art.mod[train.index,]
  test=art.mod[-train.index,]
  fit1=mfp(y~fp(x1)+x2+fp(x3)+x4a+x4b+fp(x5,df=2)+fp(x6)+fp(x7)+x8+x9a+x9b+fp(
      x10),
           data = train,family = gaussian,select=0.05,alpha = 0.05, x=TRUE)
  fit2=shrink(fit1,type="global", method = "jackknife")
  fit3=shrink(fit1, type = "parameterwise", method = "jackknife")
  if (!is.na(fit1$powers["x4a",1])&&!is.na(fit1$powers["x4b",1])){
    list_join=append(list_join,list(c("x4a.1","x4b.1")))
  }

  if (!is.na(fit1$powers["x9a",1])&&!is.na(fit1$powers["x9b",1])){
    list_join=append(list_join,list(c("x9a.1","x9b.1")))
  }
```

```
for (i in c("x1","x3","x5","x6","x7","x10")){
  if (!is.na(fit1$powers[i,2])){
    list_join=append(list_join,list(c(paste0(i,".1"), paste0(i,".2"))))
  }
}
if (length(list_join)==0){
  fit4=fit3
} else{
  fit4=shrink(fit1, type = "parameterwise", join=list_join, method = "
     jackknife" )
}
## Boosting with FP
fit5=glmboost(y~FP(x1,scaling = FALSE)+x2+FP(x3,scaling = FALSE)+x4a+x4b+
                FP(x5,p=-0.5, scaling = FALSE)+FP(x6, scaling = FALSE)+FP(x7
                    , scaling = FALSE)+
                x8+x9a+x9b+FP(x10, scaling = FALSE),
              data = train,family=Gaussian(),control = boost_control(mstop
                =2000))
cvm1=cvrisk(fit5, folds = cv(model.weights(fit5), type = "kfold", B = 10) )
fit5[mstop(cvm1)]
## MFP-Lasso
## Train data in matrix
x.lasso=fit1$x
## Create test data in matrix according to final model
test.lasso=matrix(1,nrow=dim(test)[1])
## Adding the  variables included in fit1 and transform those in
   corresponding FP forms.
for (i in row.names(fit1$powers)){
  p1=fit1$powers[i,1]
  p2=fit1$powers[i,2]
  c=fit1$scale[i,2]
  if (!is.na(p1)&&!is.na(p2)){
    if (p1==p2){
      if (p1==0){
        test.lasso=cbind(test.lasso,log(test[i]/c),(log(test[i]/c))^2)
      } else{
        test.lasso=cbind(test.lasso, (test[i]/c)^p1,(test[i]/c)^p1*log(test[
           i]/c))
      }
    }else{
      if (p1==0){
        test.lasso=cbind(test.lasso,log(test[i]/c),(test[i]/c)^p2)
      }else if(p2==0){
        test.lasso=cbind(test.lasso,(test[i]/c)^p1,log(test[i]/c))
      }else{
        test.lasso=cbind(test.lasso,(test[i]/c)^p1,(test[i]/c)^p2)
      }
    }
  } else if (!is.na(p1)){
    if(p1==0){
      test.lasso=cbind(test.lasso,log(test[i]/c))
```

```
81        }else{
82          test.lasso=cbind(test.lasso, (test[i]/c)^p1)
83        }
84      }
85    }
86    for (i in row.names(fit1$powers)){
87      if (is.na(fit1$powers[i,1])){
88        x.lasso=cbind(x.lasso,train[i])
89        test.lasso=cbind(test.lasso,test[i])
90      }
91    }
92    x.lasso=as.matrix(x.lasso)
93    y.lasso=as.matrix(train["y"])
94    cv.lasso=cv.glmnet(x=x.lasso,y=y.lasso,nfolds = 10, alpha = 1)
95    fit6=glmnet(x=x.lasso,y=y.lasso,lambda = cv.lasso$lambda.min,alpha = 1,
          family = "gaussian")
96    ## MFP-boosting
97    formula7<-as.formula(paste0(as.character(fit1$formula)[2],as.character(fit1$
          formula)[1],
98                                as.character(fit1$formula)[3],'+',
99                                paste0(rownames(fit1$trafo)[is.na(fit1$trafo)],
                                      collapse = '+')))
100   fit7=glmboost(formula7, data=train, family = Gaussian(), control = boost_
          control(mstop = 2000))
101   cvm2=cvrisk(fit7, folds = cv(model.weights(fit7), type = "kfold", B = 10))
102   fit7[mstop(cvm2)]
103   ## Compute MSPE and MSE
104   test.error[j,1]=mean((test$y-predict(fit1, newdata = test, type = "response"
          ))^2)
105   sq.residual[j,1]=mean((train$y-predict(fit1,newdata = train, type="response"
          ))^2)
106   test.error[j,2]=mean((test$y-predict(fit2, newdata = test, type = "response"
          ))^2)
107   sq.residual[j,2]=mean((train$y-predict(fit2,newdata=train,type="response"))
          ^2)
108   test.error[j,3]=mean((test$y-predict(fit3, newdata = test, type = "response"
          ))^2)
109   sq.residual[j,3]=mean((train$y-predict(fit3,newdata=train,type="response"))
          ^2)
110   test.error[j,4]=mean((test$y-predict(fit4, newdata = test, type = "response"
          ))^2)
111   sq.residual[j,4]=mean((train$y-predict(fit4,newdata=train,type="response"))
          ^2)
112   test.error[j,5]=mean((test$y-predict(fit5[mstop(cvm1)], newdata = test, type
          = "response"))^2)
113   sq.residual[j,5]=mean((train$y-predict(fit5[mstop(cvm1)],newdata = train,
          type="response"))^2)
114   test.error[j,6]=mean((test$y-predict(fit6, newx = as.matrix(test.lasso),type
          ="response"))^2)
115   sq.residual[j,6]=mean((train$y-predict(fit6,newx= x.lasso,type="response"))
          ^2)
```

```
  test.error[j,7]=mean((test$y-predict(fit7, newdata = test, type ="response"))^2)
  sq.residual[j,7]=mean((train$y-predict(fit7,newdata = train, type ="response"))^2)
}

# Boxplots of MSPE
par(mfrow=c(1,2),oma=c(5,0,0,0),cex.lab=1.4,cex.axis=1.4,las=2)
boxplot(test.error[,-5],ylim=c(0.4,0.8),xaxt="n")
title(ylab = "Mean squared prediction error")
axis(1,at=1:6,
     label=c("MFP","MFP with Global","MFP with PSF","MFP with JSF","MFP-Lasso","MFP-Boosting"))
boxplot(test.error[,5],ylim=c(0.4,5))
axis(1,at=1,label="Boosting with FP")
title(ylab = "Mean squared prediction error")

# Boxplots of difference in MSPE
par(mfrow=c(1,2),oma=c(5,0,0,0),cex.lab=1.4,cex.axis=1.4,las=2)
boxplot(test.error[,-5]-test.error[,1],ylim=c(-0.1,0.05),xaxt="n")
title(ylab = "Mean squared prediction error")
abline(h=0,lty=2,col="red")
axis(1,at=1:6,
     label=c("MFP","MFP with Global","MFP with PSF","MFP with JSF","MFP-Lasso","MFP-Boosting"))
boxplot(test.error[,5]-test.error[,1],ylim=c(0,10))
title(ylab = "Mean squared prediction error")
abline(h=0,lty=2,col="red")
axis(1,at=1,label="Boosting with FP")
```