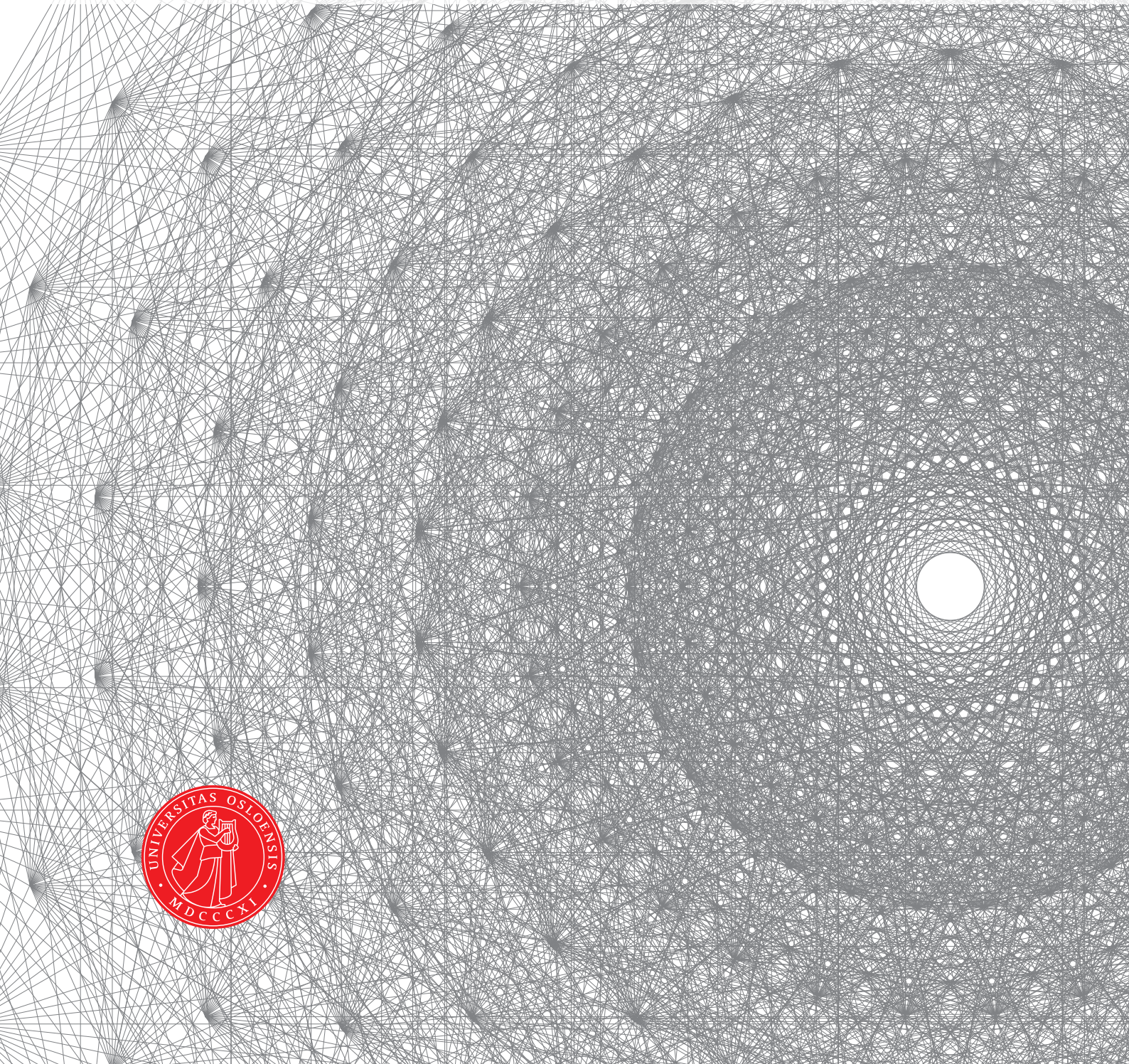
UiO : Department of Mathematics
University of Oslo

Priority boosting and Lasso-based block boosting: two novel approaches for multi-omics analysis

Yoshimi Kujime

Master's Thesis, Spring 2022



This master's thesis is submitted under the master's programme *Stochastic Modelling, Statistics and Risk Analysis*, with programme option *Statistics*, at the Department of Mathematics, University of Oslo. The scope of the thesis is 60 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.

Abstract

Developments in high-throughput technology have made multi-omics data available on a large scale. Multi-omics data are datasets consisting of different types of high-dimensional molecular variables, such as transcriptomic, proteomic, and methylation data. In recent decades, predictive modeling incorporating different types of data has attracted much attention. This thesis presents two novel boosting approaches to build a regression model for high-dimensional data consisting of multiple groups of variables such as multi-omics data. One method is *priority boosting* and the other is *Lasso-based block boosting*.

Priority boosting processes data in a hierarchical manner by setting the priority order among groups, which builds a model incorporating prior knowledge and/or practical constraints. On the other hand, Lasso-based block boosting (hereinafter called LBboost) does not have a hierarchical structure. In this method, fitting will be performed in each group separately at each boosting round, and iteratively updates the model via comparing the estimates by each group and selecting the group that gives the best update, by what we call the subset-updating approach. Both priority boosting and LBboost have several desirable properties especially in the high-dimensional setting, such as automated variable selection, shrinkage of estimates and interpretability of the resulting models.

We applied these two methods on simulation data and a real multi-omics dataset, and compared their prediction performances with three other methods, priority-Lasso, Lasso and componentwise gradient boosting (glmboost). Priority boosting tended to provide sparser prediction models that favor predictors in blocks with higher priorities over predictors in blocks with lower priorities. These results suggest that priority boosting can be regarded as a practical method that is easy to apply and interpret. On the other hand, the resulting models of LBboost tended to be less sparse than the other methods. However, in terms of the prediction accuracy, it showed relatively good results. It could often reach better or similar prediction accuracy compared to the priority boosting and priority-Lasso in our datasets. Furthermore, the results show that LBboost works well in the situations where glmboost and Lasso are prone to be overfitting.

Acknowledgements

First and foremost, I would like to thank my supervisor, Riccardo De Bin, for introducing me to this exciting topic, providing guidance throughout and giving me ideas and feedback. Thank you so much for all the hours spent on meetings and for your helpful advice.

I also would like thank to Tobias Herold and the AMLCG for providing the data. With the support of many people, I was able to write this thesis and lead a fulfilling student life. There are too many to name, but I am grateful to all the wonderful people who I have met at Blindern. I had a great time in this nice environment.

Last but not least, I am grateful to my friends and family. Thank you for both meaningful discussions and meaningless jokes, as well as for your support and encouragements.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Recent technological developments have brought many data-intensive disciplines. A conspicuous instance is biomedical science, where high-throughput technologies have made it possible to obtain *omics data*, i.e., data stemming from molecular processes on a large scale. A characteristic of omics data is a sheer number of explanatory variables (predictors), which often have more than 1 000, 10 000 or even 100 000 variables, but a smaller sample size such as several decades or hundreds. Data in which the number of predictors p is larger than the number of observations n are termed as high-dimension low-sample-size data or simply *high-dimensional data*. In this setting, it is known that most conventional estimation methods for regression break down (Mayr et al., 2014). Methods used to build prediction models for omics data require the ability to handle data in the $p > n$ setting.

In addition to the $p > n$ problem, this place demands the integration of different types of data into a prediction model. The derivation of regression models where different data sources are available is one of the most recent challenges in applied statistics. In the context of handling omics data, there are different types of high-throughput molecular variables are available, such as transcriptomic, proteomic, and methylation data. Omics data consisting of different groups of variables are called *multi-omics data*. In recent decades, prediction modelling incorporating a single type of omics data has been a well-studied topic, and further several methods to handle multi-omics data have been proposed (Herrmann et al., 2020).

One strategy to build prediction models with data consisting of multiple groups of variables is processing the data in a hierarchical manner; It assigns priorities to each of the groups (blocks) and process data according to the priority order. This strategy was introduced by Klau et al. (2018) as the principle of *priority-Lasso*. The process starts from the group with the highest priority, which is the group consisting of "favorite" variables for some reasons. Once the information from the block has been exploited, the remaining blocks are processed in turn, until the block with the lowest priority. For example, variables that are easy to measure or expected to have high prediction accuracy are assumed as favorites. Via the priority order among blocks, this modelling incorporates the user's preference. The resulting model may be regarded as a compromise between "what the data tell us" and "prior knowledge and/or realistic constraints in practice" (Klau et al., 2018).

In this thesis, we present two novel approaches to process high-dimensional

data consisting of multiple groups of variables such as multi-omics data. One method is *priority boosting* and the other is *Lasso-based block boosting*. As the names indicates, while priority-Lasso is a method based on Lasso regression, priority boosting is based on *boosting*. Priority boosting was devised seeking to combine the above strategy in priority-Lasso and boosting algorithms.

Boosting is one of the advanced methods for data analysis. It was advent as a powerful prediction method for classification in the machine learning community, and it has extended to deal with statistical problems, triggering a lot of research in the recent decades. In the context of regression problems, boosting has several advantages. Some of boosting approaches are applicable and stable in the $p > n$ setting (Hastie et al., 2009). They incorporate automated variable selection and shrinkage of estimates. These are desirable characteristics to have a sparse model and to avoid overfitting. Further, their resulting models have the form of the generalized additive model (GAM), which is interpretable in definition. Interpretability often gives us insight into the data. The aim of priority boosting is to build a prediction model that incorporates advantages from both boosting and the hierarchical strategy in priority-Lasso.

On the other hand, Lasso-based block boosting (hereinafter called LBboost) does not take the hierarchical strategy. This method processes high-dimensional data containing multiple groups, but does not assign priorities to each of the groups. After dividing variables in data into several groups, LBboost processes the data, updating the effect of predictors in each blocks separately which is purely data-driven. This method is useful in the cases where there is no reasonable basis for setting priorities among groups. We do not have an opportunity to use prior knowledge unlike priority-Lasso or priority boosting, but can listen to what the data tell us more clearly.

The main purpose of this thesis is to introduce the two methods, priority boosting and LBboost, and evaluate the prediction performances on simulation data and real data. This thesis is structured as follows.

In Chapter 2, we review background materials in statistical methodologies for the following chapters. We start with boosting. One of the most prominent boosting algorithms, the gradient boosting is mainly described. Next, survival analysis is covered. A Boosting algorithm in the cases where responses are survival times is explained there. We then discuss Lasso and priority-Lasso.

In Chapter 3, we present the novel approaches: priority boosting in Section 3.1 and LBboost in Section 3.2. In each section, we first provide a brief introduction in subsection "Principles", which gives key characteristics of the method and a succinct overview of its algorithm. Then, in subsection "Algorithm", we introduce the algorithm formally.

We implemented these methods in R functions `priorityboost` and `LBboost`, which used in analysis in Chapter 4 and Chapter 5. In Chapter 4, we conduct a simulation study to evaluate the prediction performances of priority boosting and LBboost. Simulation design has two parts; one part is for Gaussian responses and the other is for survival times. Their performances are compared with three other methods, priority-Lasso, Lasso and gradient boosting.

In Chapter 5, we asses the two methods' prediction abilities on real multi-omics data, the responses which of are survival times. Their prediction

performances are compared again with the three methods, priority-Lasso, Lasso and gradient boosting.

Finally, in Chapter 6, we present our conclusion summarizing our findings and ideas for future work. In addition, we provide two appendices. Appendix A presents the R implementation for priority boosting and LBboost. Appendix B shows the R codes to generate the simulation data that we discussed in Chapter 4.

CHAPTER 2

Preliminaries

2.1 Boosting

Boosting is referred to as one of the most promising methodological approaches for data analysis developed in the past few decades (Mayr et al., 2014). Boosting has its roots in the machine learning field, and evolved from a black-box algorithm to a flexible method to construct interpretable statistical models. The main aim of this section is to introduce gradient boosting, starting with a short historical overview. In Section 2.1.4, the componentwise approach is discussed, which works for high-dimensional data where the number of predictors exceeds the number of observations. With the tuning parameters, how shrinkage and automated variable selection can be done in the componentwise approach is described in Section 2.1.5. Boosting for survival analysis is covered in Section 2.2.4.

2.1.1 AdaBoost: answer to question in machine learning

Kearns & Valiant (1989) posed a question: Could any "weak" learning algorithms for classification be converted into a "strong" learner? A weak learner may be considered as a method that predicts slightly better than random guess. In binary classification, its accuracy would be just better than a coin flip. A strong learner, by contrast, should be able to perform a nearly perfect prediction. Answering to this question, Schapire (1990) and Freund (1990) introduced the first boosting algorithms. While these were rather theoretical constructs than tools intended for practical use, in 1997, a concrete solution for binary classification task was developed in the form of AdaBoost, the first practical boosting algorithm (Freund & Schapire, 1996).

The basic idea of boosting is to iteratively train with simple learning algorithms, called *weak base learners*, and then combine those solutions to improve prediction accuracy. In the iterations, one does not manipulate the base learner itself, but the training data. In the AdaBoost algorithm, the data modification at each boosting iteration is performed by applying weight to each observation. The base learner (base procedure) G is sequentially applied to the data modified at each iteration, to obtain a sequence of fitted weak learners $G^{[m]}$ for $m = 1, \ldots, M$.

Assume that we have a dataset $(\boldsymbol{x}_i, y_i)$ for $i = 1, \ldots, n$, where n is the sample size, $\boldsymbol{x}_i \in \mathbb{R}^p$ is a p-dimensional predictor vector, and $y_i \in \{-1, 1\}$ is a binary response. A classifier $G(\boldsymbol{x}_i)$ produces a prediction taking one of the two

values $\{-1, 1\}$. Firstly, all the weights w_i are initialized as $1/n$. Since these are the same values for all observations, at the first iteration, the base learner $G^{[1]}$ are simply fitted on the training data in the usual manner. In each iteration $m = 1, 2, \ldots, M$, we compute the following *error rate*,

$$err^{[m]} = \frac{\sum_{i=1}^{n} w_i I(y_i \neq G^{[m]}(\boldsymbol{x}_i))}{\sum_{i=1}^{n} w_i} \tag{2.1.1}$$

and the iteration-specific coefficient,

$$\alpha^{[m]} = \log \frac{1 - err^{[m]}}{err^{[m]}}, \tag{2.1.2}$$

which shows the classification performance of the weak learner $G^{[m]}$. With this coefficient, the weights w_i are updated to $w_i \exp\{\alpha^{[m]} I(y_i \neq G^{[m]}(\boldsymbol{x_i}))\}$. The weights for the observations that were misclassified at the previous iteration are increased, while the weights for those that were correctly classified are not.

The final classifier G_{Adaboost} is expressed as the weighted sum of all the weak learners $G^{[m]}$,

$$G_{\text{Adaboost}} = \text{sign}\left[\sum_{m=1}^{M} \alpha^{[m]} G^{[m]}(\boldsymbol{x})\right]. \tag{2.1.3}$$

From equation (2.1.2), we can see that the coefficient $\alpha^{[m]}$ becomes smaller as the error rate $err^{[m]}$ becomes larger, so that the base classifier $G^{[m]}$ with smaller error rate will have a greater impact on the final classification compared to classifiers with larger error rate. For a schematic overview of Adaboost, see Algorithm 2.1.1.

By increasing the weights of the observations that were misclassified in the previous round, the algorithm keeps shifting the focus on observations that are difficult to predict. This leads to dramatically improved performance compared to training by a single base learner such as simple classification trees or stumps. The introduction of AdaBoost attracted a great deal of attention in the machine learning community. Breiman, a pioneer in machine learning, referred to AdaBoost as the "best off-the-shelf classifier in the world" (Hastie et al., 2009).

2.1.2 From black box to statistical modeling

Adaboost was originally regarded as a black-box algorithm like most of machine learning algorithms. Friedman et al. (2000) provided a statistical perspective on Adaboost by showing that this algorithm indeed minimizes the exponential loss $L(y, f(x)) = \exp(-yf(x))$ via the *forward stagewise additive modeling*. The forward stagewise additive modeling is a method to seek an additive expansion in a set of basis functions that minimizes the *empirical risk*. An additive expansion is expressed as

$$\sum_{m=1}^{M} \beta_m b(\boldsymbol{x}; \boldsymbol{\gamma}_m), \tag{2.1.4}$$

Algorithm 2.1.1 Adaboost

1. Initialize the observation weights $w_i = 1/n$ for $i, \ldots, n$.
2. For $m = 1, \ldots, M$
   1) Fit classifier $G^{[m]}(\boldsymbol{x})$ to the training data with weights w_i.
   2) Compute error rate,

$$err^{[m]} = \frac{\sum_{i=1}^{n} w_i I(y_i \neq G^{[m]}(\boldsymbol{x}_i))}{\sum_{i=1}^{n} w_i}$$

   3) Compute the iteration-specific coefficient,

$$\alpha^{[m]} = \log \frac{1 - err^{[m]}}{err^{[m]}},$$

   4) Update individual weights

$$w_i \leftarrow w_i \cdot \exp\Big\{\alpha^{[m]} I(y_i \neq G^{[m]}(\boldsymbol{x}_i))\Big\}.$$

3. Compute the final classifier

$$G_{\text{Adaboost}}(\boldsymbol{x}) = \text{sign}\left[\sum_{m=1}^{M} \alpha^{[m]} G^{[m]}(\boldsymbol{x})\right].$$

where β_m, for $m = 1, \ldots, M$, are the expansion coefficients and $b(\boldsymbol{x}; \boldsymbol{\gamma}_m)$ are basis functions of the multivariate argument $\boldsymbol{x}$ with parameters $\boldsymbol{\gamma}_m$. The emperical risk is a loss function averaged over a training dataset, $(\boldsymbol{x}_i, y_i)$ for $i, \ldots, n$

$$\hat{f} = \underset{f}{\text{argmin}} \left\{\frac{1}{n} \sum_{i=1}^{n} L(y_i, f(\boldsymbol{x}_i))\right\}. \tag{2.1.5}$$

The forward stagewise additive modeling approximately solves the optimization problem,

$$\hat{f} = \underset{f}{\text{argmin}} \sum_{i=1}^{n} L\left(y_i, \sum_{m=1}^{M} \beta_m b(\boldsymbol{x}; \boldsymbol{\gamma}_m)\right). \tag{2.1.6}$$

Generally, solving this problem requires computationally intensive numerical optimization techniques, but the forward stagewise additive modeling can be a reasonable alternative. This algorithm is a simple iterative method by sequentially adding new basis function to the current expansion without adjusting the previously added terms, which provides an approximation of the solution. For a detailed explanation of the equivalence between AdaBoost and the forward stagewise additive modeling with the exponential loss function, we refer to Hastie et al. (2009). Mayr et al. (2014) emphasized that this interpretation by Friedman et al. (2000) is the most important view in the context of the evolution of boosting from a black-box algorithm to a statistical

method. This work provided the basis for understanding boosting in a statistical framework.

2.1.3 Gradient boosting

The statistical view of boosting led to the extension of the principle of boosting to regression problems. A regression model aims at quantifying the relation between the predictors and the response through an interpretable function $f(\boldsymbol{x}) = E(Y|\boldsymbol{X} = \boldsymbol{x})$, known as the *regression function*, where $Y \in \mathbb{R}$ denotes a random output variable, $\boldsymbol{X} \in \mathbb{R}^p$ a p-dimensional random input vector and $\boldsymbol{x}$ is a particular realization of $\boldsymbol{X}$.

In general, to estimate the regression function, we seek a function that minimizes the expectation of a loss function L,

$$\hat{f} = \underset{f}{\operatorname{argmin}} \left\{ E_{\boldsymbol{X},Y}[L(Y, f(\boldsymbol{x}))] \right\}. \tag{2.1.7}$$

In practice, we minimize the empirical risk,

$$\hat{f} = \underset{f}{\operatorname{argmin}} \left\{ \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(\boldsymbol{x}_i)) \right\}. \tag{2.1.8}$$

Friedman (2001) defined one of the most prominent statistical boosting approaches, gradient boosting. Gradient boosting estimates f by iteratively updating its value, to (approximately) minimizes the empirical risk. As the solution of gradient boosting, we obtain the estimate of f as the form $\hat{f} = C + \sum_{m=1}^{M} h^{[m]}$, where C is an initial guess of f and $h^{[m]}$ are the increments called *step*, which improve the estimate iteratively.

To understand the gradient boosting algorithm, analogy to steepest descent method may be useful. In the following, gradient boosting and functional gradient descent (FGD) are used as equivalent terminology for the same method (Algorithm 2.1.2). Steepest descent is a well-known iterative method for minimization of a function. Consider that we would like to find the minimum of a function $g(\boldsymbol{x})$, where $\boldsymbol{x} \in \mathbb{R}^p$ and $g : \mathbb{R}^p \to \mathbb{R}$. Let $g'(\boldsymbol{x})$ denote the gradient of g evaluated point $\boldsymbol{x}$. The opposite direction of $g'(\boldsymbol{x})$ indicates the steepest direction downhill on the surface of g at the point $\boldsymbol{x}$. The method minimizes g by repeatedly taking a step in the direction of the negative gradient of g. The updating equation is expressed as

$$\boldsymbol{x}^{[m+1]} = \boldsymbol{x}^{[m]} - \nu^{[m]} g'(\boldsymbol{x}^{[m]}).$$

where $\nu^{[m]} > 0$.

FGD algorithm has similar structure to steepest descent method, but for the directions of the steps, FGD does not use the negative gradient itself. The negative gradient of the objective function, i.e., a specific loss function, is defined only at the data points in a training set. Since the aim of FGD is to build a model for any input variables $\boldsymbol{x} \in \mathbb{R}^p$, we want to have a generalized version of the negative gradient that is defined in any $\boldsymbol{x}$ values other than the training sample points. One way to do is to fit base learners to the negative gradient.

As an example of the base learners, consider a multivariate linear model fitted by least squares. Let the negative gradient vector in the mth iteration

be denoted by $\boldsymbol{u}^{[m]} = (u_1^{[m]}, \ldots, u_n^{[m]})$. The component $u_i^{[m]}$ for $i = 1, \ldots, n$ is expressed as

$$u_i^{[m]} = - \left. \frac{\partial L(y_i, f(\boldsymbol{x}_i))}{\partial f(\boldsymbol{x}_i)} \right|_{f(\boldsymbol{x}_i)=\hat{f}^{[m-1]}(\boldsymbol{x}_i)}, \tag{2.1.9}$$

where $\boldsymbol{x}_i \in \mathbb{R}^p$ is a p-dimensional input vector. With the base learner, we estimate $\hat{\boldsymbol{\beta}}^{[m]} = (X^T X)^{-1} X \boldsymbol{u}^{[m]}$, where X is $n \times p$ matrix, whose component X_{ij} represents the jth predictor for the ith sample. Estimating $\hat{\boldsymbol{\beta}}^{[m]}$ means to draw a regression surface using the data points in a training set, which shows a relation between $\boldsymbol{u}^{[m]}$ and X. Then, for any $\boldsymbol{x} \in \mathbb{R}^p$, the fitted base learner will be defined as $\hat{b}^{[m]} = \boldsymbol{x}\hat{\boldsymbol{\beta}}^{[m]}$. $\hat{b}^{[m]}$ may be regarded as a function to give an approximation of $\boldsymbol{u}^{[m]}$ for $\boldsymbol{x} \in \mathbb{R}^p$, where $\boldsymbol{x}$ need not be a value in the training set. Using this function in each step, FGD iteratively improves the estimate.

The updating equation in FGD becomes

$$\hat{f}^{[m]} = \hat{f}^{[m-1]} + \nu \hat{b}^{[m]}, \tag{2.1.10}$$

and the final estimate is expressed as

$$\hat{f}^{[m_{stop}]} = \hat{f}^{[0]} + \nu \sum_{m=1}^{m_{\text{stop}}} \hat{b}^{[m]}. \tag{2.1.11}$$

FGD has two tuning parameters; ν and m_{stop}. The parameter $0 < \nu < 1$ is called *boosting step size* or *step-length factor*, and controls the learning rate of the boosting procedure. The stopping iteration m_{stop}, which is the main tuning parameter, is often chosen by evaluating the models with cross-validation or bootstrapping. Too large m_{stop} may lead to overfitting of the final estimate. More details on these tuning parameters will be discussed in section 2.1.5.

In this algorithm, we can apply any regression technique as the base learners. Common choices are the linear least-squares model with $b = \boldsymbol{x}\boldsymbol{\beta}^{[m]}$, smoothing splines or stumps. Also any loss function that is convex and differentiable can be used. An example is the L_2 loss, $L(y_i, f) = \frac{1}{2}(y - f)^2$, which is called also squared error loss. L_2Boosting, the gradient boosting with L_2 loss, is the simplest and one of the most used boosting algorithm. The negative gradient of the L_2 loss is equivalent to the residual, $y - \hat{f}$. Thus, in each boosting iteration m, L_2Boosting fits the base learner to the residuals from the previous iteration, $y - \hat{f}^{[m-1]}$. Here, we can find a common concept for gradient boosting and Adaboost; both boosting algorithms improve their performance by focusing on observations that are difficult to predict in the previous update. While AdaBoost gives higher weight to observations that were misclassified before, L_2Boosting iterates refitting the residuals given previous estimates. More details on L_2Boosting can be found in Bühlmann & Hothorn (2007).

2.1.4 Componentwise boosting

The componentwise boosting is an effective method for building models for high-dimensional data where the number of input variables exceeds the number of observations. This algorithm incorporates variable selection during model estimation. The fundamental concept of the componentwise boosting is to update the effect of only one predictor at a time, instead of updating the

Algorithm 2.1.2 generic FGD algorithm

1. Set iteration counter $m = 0$. Specify loss function $L(y_i, f)$. Initialize the estimate $\hat{f}^{[0]}$. Common choices are $\hat{f}^{[0]} = 0$ or

$$\hat{f}^{[0]} = \underset{c}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, c).$$

2. For $m = 1, \ldots, m_{stop}$,
   1) Compute the negative gradient of the loss function evaluated in the previous iteration.
   $$u_i^{[m]} = - \frac{\partial}{\partial f} L(y_i, f) \Big|_{f=\hat{f}^{[m-1]}(\boldsymbol{x}_i)} \quad \text{for } i = 1, \ldots, n.$$
   2) Fit the negative gradient vector $\boldsymbol{u} = (u_1, \ldots, u_n)$ to $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ by the base learner (base procedure), giving the fitted base learner $\hat{b}^{[m]}$
   $$(\boldsymbol{x}_i, u_i)_{i=1}^n \overset{\text{base learner}}{\rightarrow} \hat{b}^{[m]}$$
   3) Update the estimate $\hat{f}^{[m]} = \hat{f}^{[m-1]} + \nu \cdot \hat{b}^{[m]}$, where $0 < \nu < 1$.
3. The final estimate is $\hat{f}^{[m_{stop}]} = \hat{f}^{[0]} + \nu \sum_{m=1}^{m_{stop}} \hat{b}^{[m]}$.

effect of all predictors. The componentwise method can be applied to different boosting algorithms, but in this section, we will only consider the componentwise version of the gradient boosting.

The componentwise gradient boosting was firstly introduced by Bühlmann & Yu (2003). Their concept was to apply componentwise base learners to the functional gradient descent (FGD). Assume that we have a training set $(\boldsymbol{x}_i, y_i)$ for $i, \ldots, n$, where $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{ip}) \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$. Setting an initial guess $f^{[0]}$, computing the negative gradient $\boldsymbol{u}^{[m]}$ of a loss function in each iteration, $m = 1, \ldots, m_{\text{stop}}$, is the same as the procedures in FGD. Then, in both the componentwise gradient boosting and FGD, we approximate the negative gradient, but the way of approximation makes difference between them. While FGD fits $\boldsymbol{u}^{[m]}$ to a single base learner, the componentwise boosting fits $\boldsymbol{u}^{[m]}$ to separate base learners corresponding to each component (each dimension) of the predictors respectively. Then we obtain p fitted base learners $\hat{b}_j^{[m]}$ for $j = 1, \ldots, p$ at each iteration m.

As an example of the base learners, consider a univariate linear model fitted by least squares, and let $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)$ denote the coefficients of the predictors. The estimate of the coefficient β_1, for example, will be obtained as

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n x_{i1} u_i}{\sum_{i=1}^n {x_{i1}}^2}. \tag{2.1.12}$$

Then, for a p-dimensional predictor vector $\boldsymbol{x} = (x_1, \ldots, x_p)$, the fitted base learner $\hat{b_1}^{[m]}$ will be defined as $\hat{b_1}^{[m]}(x_1) = \beta_1 x_1$. Similarly, we obtain $\hat{b}_2^{[m]}, \ldots, \hat{b}_p^{[m]}$.

Among them, we select the base learner that fits $\boldsymbol{u}^{[m]}$ the best. We usually choose the base learner $\hat{b}_{j^*}^{[m]}$ that minimizes the most the empirical risk, where

$$j^* = \underset{j \in \{1,\ldots,p\}}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \hat{f}^{[m-1]} + \hat{b}_j^{[m]}). \tag{2.1.13}$$

Setting that $\hat{b}^{[m]} = \hat{b}_{j^*}^{[m]}$, the updating equation becomes

$$\hat{f}^{[m]} = \hat{f}^{[m-1]} + \nu \hat{b}^{[m]} \tag{2.1.14}$$

where $0 < \nu < 1$ is the step-length factor. The final estimation is

$$\hat{f}^{[m_{\text{stop}}]} = \hat{f}^{[0]} + \nu \sum_{m=1}^{m_{\text{stop}}} \hat{b}^{[m]}. \tag{2.1.15}$$

A schematic overview of the componentwise gradient boosting can be found in Algorithm 2.1.3.

Here, we introduce one of the most important model classes, *generalized additive model* (GAM), where the conditional distribution of the response is assumed to follow an exponential family distribution. GAM has the form,

$$g(E(Y|\boldsymbol{x})) = \beta_0 + h_1(x_1) + h_2(x_2) + \cdots + h_p(x_p). \tag{2.1.16}$$

Here, g is a *link* function that is strictly increasing and differentiable. β_0 is an intercept and $h_1(x_1), \ldots, h_p(x_p)$ are the effects of covariates $x_1, \ldots, x_p$ on the transform of the expectation of the response. The right hand side of the equation, $\beta_0 + h_1(x_1) + h_2(x_2) + \cdots + h_p(x_p)$, is denoted by $\eta(\boldsymbol{x})$. We highlight that $\eta(\boldsymbol{x})$, which is called *additive predictor*, is interpretable. The partial effect of each covariate x_j for $j = 1, \ldots, p$ on the response is represented by function h_j respectively. Based on h_j, the direction, size and shape of the effect of x_j can be interpreted.

When the conditional distribution of the response is assumed to follow an exponential family distribution, the componentwise gradient boosting fits GAM where the link function g is the identity link,

$$E(Y|\boldsymbol{x}) = \beta_0 + h_1(x_1) + h_2(x_2) + \cdots + h_p(x_p), \tag{2.1.17}$$

which is often called additive model. Note that componentwise gradient boosting can be used even though the distribution is not assumed to follow an exponential family distribution. The resulting models have the same form as the additive predictor in GAM and keep the interpretability. For more details on GAM, we refer to Hastie & Tibshirani (1990).

The componentwise boosting algorithm may be slightly modified to update the parameter estimate, instead of updating the function estimate. Assume that we have a univariate least-squares estimator as the base learner and at each iteration, we compute $\hat{\gamma}_j = \sum_{i=1}^{n} x_{ij} u_i / \sum_{i=1}^{n} {x_{ij}}^2$ for $j = 1, \ldots, p$, where $\hat{\gamma}_j$ denotes the estimate of the coefficient of the linear model that is the fitted base

Algorithm 2.1.3 Componentwise gradient boosting

1. Set iteration counter $m = 0$. Specify a loss function $L(y_i, f)$ and a set of base learners $b_1, \ldots, b_p$. Initialize the estimate $\hat{f}^{[0]}$. Examples of the starting value are $\hat{f}^{[0]} = 0$ or
$$\hat{f}^{[0]} = \underset{c}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, c).$$
2. For $m = 1, \ldots, m_{\text{stop}}$,
   1) Compute the negative gradient of the loss function evaluated at the previous iteration.
   $$u_i^{[m]} = -\frac{\partial}{\partial f} L(y_i, f)\Big|_{f=\hat{f}^{[m-1]}(\boldsymbol{x}_i)} \quad \text{for } i = 1, \ldots, n.$$
   2) Fit the negative gradient vector $\boldsymbol{u}^{[m]} = (u_1^{[m]}, \ldots, u_n^{[m]})$ separately to each base learner $\hat{b}_j^{[m]}$, $j = 1, \cdots, p$,
   $$(x_{ij}, u_i)_{i=1}^{n} \overset{\text{fitting by base learner}}{\rightarrow} \hat{b}_j^{[m]},$$
   where x_{ij} denotes the jth covariate for the ith ovservation.
   3) Select the component j^* that gives the best update. A common choice is
   $$j^* = \underset{j \in \{1,\ldots,p\}}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \hat{f}^{[m-1]} + \hat{b}_j^{[m]}).$$
   Set $\hat{b}^{[m]} = \hat{b}_{j^*}^{[m]}$.
   4) Update the estimate $\hat{f}^{[m]} = \hat{f}^{[m-1]} + \nu \hat{b}^{[m]}$, where $0 < \nu < 1$ is the step-length factor.
3. The final estimate is $\hat{f}^{[m_{\text{stop}}]} = \hat{f}^{[0]} + \nu \sum_{m=1}^{m_{\text{stop}}} \hat{b}^{[m]}$.

learner. Then, we choose the best $\hat{\gamma}_{j^*}$ among them. The updating equation is expressed as

$$\hat{\beta}_j{}^{[m]} = \begin{cases} \hat{\beta}_j{}^{[m-1]} + \nu \hat{\gamma}_j{}^{[m]} & \text{if } j = j^* \\ \hat{\beta}_j{}^{[m-1]} & \text{otherwise.} \end{cases} \tag{2.1.18}$$

The final fitted linear predictor is

$$\hat{f}^{[m_{\text{stop}}]}(\boldsymbol{x}_i) = \boldsymbol{x}_i \hat{\boldsymbol{\beta}}^{[m_{\text{stop}}]}. \tag{2.1.19}$$

Algorithm 2.1.4 shows the overall procedure for the L_2Boosting with the componentwise least-squares base procedure as an example of this parameter-updating approach.

2.1.5 Early stopping and variable selection

As with FGD, in the componentwise gradient boosting, two tuning parameters, stopping iteration m_{stop} and step-length factor ν, control the shrinkage of the

Algorithm 2.1.4 L_2Boosting with componentwise least-squares base procedure

1. Set iteration counter $m = 0$. The loss function is set to L_2 loss, $\frac{1}{2}(y - f)^2$. Initialize the estimate $\hat{\boldsymbol{\beta}}^{[0]} = (\hat{\beta}_1^{[0]}, \ldots, \hat{\beta}_p^{[0]}) = (0, \ldots, 0)$ and set $\hat{f}^{[0]}(\boldsymbol{x}_i) = \hat{\eta}_i^{[0]} = \boldsymbol{x}_i \hat{\boldsymbol{\beta}}^{[0]} = 0$.
2. For $m = 1, \ldots, m_{\text{stop}}$,
   1) Compute the residual from the previous iteration,
   $$u_i^{[m]} = y_i - \hat{f}^{[m-1]}(\boldsymbol{x}_i) \text{ for } i = 1, \ldots, n.$$
   2) Fit the residual vector $\boldsymbol{u}^{[m]} = (u_1^{[m]}, \ldots, u_n^{[m]})$ separately to the base learners $\hat{b}_j^{[m]}$ that are univariate linear models for $j = 1, \cdots, p$. Obtain the coefficient estimates,
   $$\hat{\gamma_j}^{[m]} = \sum_{i=1}^{n} x_{ij} u_i / \sum_{i=1}^{n} {x_{ij}}^2,$$
   where x_{ij} is the jth predictor for the ith observation. $\hat{\gamma_j}^{[m]} x_{ij}$ can be seen as an approximation of $u_i^{[m]}$.
   3) Select the best component j^* such that
   $$j^* = \underset{j}{\operatorname{argmin}} \sum_{i=1}^{n} (u_i^{[m]} - \hat{\gamma_j}^{[m]} x_{ij})^2.$$
   4) Update the estimates for the coefficients,
   $$\hat{\beta_j}^{[m]} = \begin{cases} \hat{\beta_j}^{[m-1]} + \nu \hat{\gamma_j}^{[m]} & \text{if } j = j^* \\ \hat{\beta_j}^{[m-1]} & \text{otherwise.} \end{cases}$$
   5) Compute the fitted linear predictor,
   $$\begin{aligned} \hat{f}^{[m]}(\boldsymbol{x}_i) &= \hat{f}^{[m-1]}(\boldsymbol{x}_i) + \nu \hat{\gamma}_{j^*}^{[m]} x_{ij^*} \\ &= \boldsymbol{x}_i \hat{\boldsymbol{\beta}}^{[m]}. \end{aligned}$$
3. The final fitted linear predictor is
   $$\hat{f}^{[m_{\text{stop}}]}(\boldsymbol{x}_i) = \boldsymbol{x}_i \hat{\boldsymbol{\beta}}^{[m_{\text{stop}}]}.$$

final estimate. Shrinkage generally reduces the variance of estimates, so that enhances the stability and accuracy of predictions (Hastie et al., 2009).

There is a trade off between them; smaller values of ν requires larger value of m_{stop}. It has been empirically found that if ν is sufficiently small, the test error potentially becomes better and almost never worse. We tipically set ν to be 0.1. The parameter m_{stop} is considered the main tuning parameter. If m_{stop} is too small, learning from the training data is not enough well, resulting in underfitting. On the other hand, with large value of m_{stop}, we can usually reduce the empirical risk, but fitting the training data too much may lead to overfitting. To avoid overfitting, it is important to evaluate the model not on the training data, but on separated test data and select the optimal number of iterations according to the test error (early stopping). We usually use cross-validation (CV) or bootstrapping to choose m_{stop}.

Parameter m_{stop} also controls the sparsity of the final model. Consider the componentwise gradient boosting. As seen above, the structure of the final estimate is equivalent as the additive predictor in (2.1.16),

$$\beta_0 + h_1(x_1) + h_2(x_2) + \cdots + h_p(x_p).$$

Here, $h_j(x_j)$ corresponds to

$$\nu \sum_{m=1}^{m_{\text{stop}}} I(j^{[m]} = j^{*[m]}) \hat{b_j}^{[m]}(x_j). \tag{2.1.20}$$

Some of $h_1(x_1), \ldots, h_p(x_p)$ are equal to zero, when the corresponding base learners have not been selected. If boosting iteration stops before all the component are selected, then the resulting model incorporates variable selection. As the value of m_{stop} is smaller, we obtain a sparser model.

2.2 Survival analysis and boosting for survival data

2.2.1 Survival data

Survival analysis is the study of survival time, so called lifetime or time to event. Examples of events are the death of a laboratory animal or the relapse of a cancer patient in biostatistics, or failure of a machine part or burning out of light bulbs in engineering. In context of econometrics and sociology, time to events include duration of unemployment or time to divorce or birth of a child.

A key characteristic of survival analysis is that the data come as a mixture of complete and incomplete observations. The event of interest occurs in some individuals, but not in others. For example, some patients have a recurrence of cancer during the study period, while others do not. Some couples get divorced, whereas others remain happily married. Incomplete observations, in which the event in question does not occur, are called *censored*.

There are several types of censoring; right-censoring occurs when we only know that the survival endpoint exceeds a certain value, and left-censoring results when the endpoint is known to be below a particular value but it is unknown by how much. Right-censored survival dataset is the most common setting, and we will deal with this type of data in the following sections. In clinical studies, right-censored observations typically occur when individuals still have no events at the end of the study, when they withdraw from the study, or when they become untraceable. For further discussion on censoring, we refer to Aalen et al. (2008)

The standard setup for right censored data for the ith individual $i = 1, \ldots, n$ is the following. We denote the survival time by T_i and the potential censoring time by C_i. Then, our observations of the survival data consist of the pairs

$$(\widetilde{T}_i, d_i)_{i=1}^n,$$

where $\widetilde{T}_i = \min(T_i, C_i)$ and $d_i = 1$ if $T_i \leq C_i$, and 0 otherwise. $\widetilde{T}_i$ denotes observed time which is either the true lifetime T_i (in the case that $T_i \leq C_i$) or the censoring time C_i (in the case that $T_i > C_i$). Censoring indicator for observed time d_i indicates whether an event has occurred or not. $d_i = 1$ if we observe the true survival time, and 0 if the observation is censored.

2.2.2 Survival function and hazard function

To analyze survival data, we introduce the *survival function* $S(t)$ and the *hazard function* $\alpha(t)$. The survival function gives the probability that the event in question has not yet occurred by time t. Assume that random variable T that denotes the lifetime is absolutely continuous and it has probability density function $f(t)$ and cumulative distribution function $F(t)$. Then the survival function is defined as

$$S(t) = P(T > t) = 1 - F(t) = \int_t^\infty f(u)du. \tag{2.2.1}$$

On the other hand, the hazard function, also called *hazard rate* or *failure rate*, is defined by conditional probability. Given that the individuals have not yet experienced the event in question by time t, the probability of experiencing the

event in a small time interval $[t, t+h)$ becomes $\alpha(t)h$. The formal definition of the hazard rate is

$$\alpha(t) = \lim_{h \to 0} \frac{1}{h} P(t < T \leq t + h \mid T > t). \tag{2.2.2}$$

The *cumulative hazard function* $A(t)$ is defined as

$$A(t) = \int_0^t \alpha(u) du. \tag{2.2.3}$$

Mathematical relation between the survival function and the hazard function is straightforward to show. By equation (2.2.1), (2.2.2) and (2.2.3), we have

$$\begin{aligned} A'(t) = \alpha(t) &= \lim_{h \to 0} \frac{1}{h} P(t < T \leq t + h \mid T > t) \\ &= \lim_{h \to 0} \frac{1}{h} \frac{P(t < T \leq t + h)}{P(T > t)} \\ &= \lim_{h \to 0} \frac{1}{h} \frac{S(t) - S(t+h)}{S(t)} \\ &= -\frac{S'(t)}{S(t)}. \end{aligned} \tag{2.2.4}$$

And then,

$$\begin{aligned} \frac{d}{dt} \ln S(t) &= -\alpha(t) \\ \ln S(t) &= -\int_0^t \alpha(u) du + \ln S(0) = -\int_0^t \alpha(u) du, \end{aligned}$$

where the last equality is because $S(0)$ is 1. We obtain

$$S(t) = \exp\left\{ -\int_0^t \alpha(u) du \right\} = \exp\{-A(t)\}. \tag{2.2.5}$$

Note that since $S'(t) = -f(t)$, $\alpha(t) = f(t)/S(t)$. Also note that $\alpha(t)$ can be any non-negative function, while $S(t)$ is a non-negative function that starts at 1 and decreases over time.

From observations of survival data, we may estimate the survival function by the Kaplan-Meier estimator (Kaplan & Meier, 1958) and the cumulative hazard rate by the Nelson-Aalen estimator (Nelson, 1969, 1972; Aalen, 1978). In the following, we assume that there are no tied event times for simplicity. Assume that we have a dataset including n individuals and that K individuals have had the event in question, and $n - K$ individuals have been censored. let $T_{(1)} < T_{(2)} < \cdots < T_{(K)}$ be the ordered times when an occurrence of the event is observed. If an individual have not experienced the event before a given time t, and have not been censored before time t, then we say that the individual is *at risk* at time t. The set of individuals who are at risk is termed the *risk set*. Let $Y(t)$ denote the number of individuals who are at risk "just before" time t and hence, might have the event at time t. Then, the Kaplan-Meier estimator is expressed as

$$\hat{S}(t) = \prod_{T_{(k)} \leq t} \left(1 - \frac{1}{Y(T_{(k)})} \right), \tag{2.2.6}$$

and the Nelson-Aalen estimator is

$$\hat{A}(t) = \sum_{T_{(k)} \leq t} \frac{1}{Y(T_{(k)})}. \tag{2.2.7}$$

These are right-continuous step functions with jumps at observed event times $T_{(k)}$. The hazard rate $\alpha(t)$ may be estimated by the "slope" of the plot of the Nelson-Aalen estimator. More details on these estimators including formal derivations can be found in Aalen et al. (2008).

2.2.3 Cox's regression model

Clinical and epidemiological studies often assess the effect of many covariates on survival in order to gain knowledge from a heterogeneous group of cases. The most commonly used regression model for censored survival data is the Cox's proportional hazards model (Cox, 1972)

In this model, it is assumed that the hazard rate $\alpha(t)$ for an individual with covariates $x_1, \ldots, x_p$ takes the form

$$\alpha(t|x_1, \ldots, x_p) = \alpha_0(t) \exp\{\beta_1 x_1 + \cdots + \beta_p x_p\}. \tag{2.2.8}$$

Here $\exp\{\beta_1 x_1 + \cdots + \beta_p x_p\}$ is called the *relative risk function* or *hazard ratio*, and $\beta_1, \ldots, \beta_p$ are regression coefficients showing the effect of the covariates. $\alpha_0(t)$ is the *baseline hazard rate* that gives the shape of the hazard rate as a function of time.

Consider two individuals, individual 1 and individual 2, with covariates $x_{i1}, \ldots, x_{ip}$ for $i = 1$ and 2. Assume all covariates are fixed over time. Then, the ratio of their hazard rates is

$$\begin{aligned} \frac{\alpha(t|x_2)}{\alpha(t|x_1)} &= \frac{\alpha_0(t) \exp\{\beta_1 x_{11} + \cdots + \beta_p x_{1p}\}}{\alpha_0(t) \exp\{\beta_1 x_{21} + \cdots + \beta_p x_{2p}\}} \\ &= \frac{\exp\{\beta_1 x_{11} + \cdots + \beta_p x_{1p}\}}{\exp\{\beta_1 x_{21} + \cdots + \beta_p x_{2p}\}}, \end{aligned} \tag{2.2.9}$$

which is constant over time. It shows that in the Cox model with fixed covariates, the ratio of the hazard rates is assumed to be constant over time and it is called the *proportional hazards assumption*.

Because of the nonparametric nature of the baseline hazard, we cannot use ordinary likelihood methods to estimate the regression coefficients. Instead we use a *partial likelihood*, which we will introduce. Consider n individuals that have covariate vectors $\boldsymbol{x}_i$ for $i = 1, \ldots, n$. Assume that K individuals out of the n have experienced the event and let $T_{(1)} < T_{(2)} < \cdots < T_{(K)}$ be the ordered times when an occurrence of the event is observed. $\boldsymbol{x}_{(1)} < \boldsymbol{x}_{(2)} < \cdots < \boldsymbol{x}_{(K)}$ denote the covariate vectors for the corresponding individuals; the individual with $\boldsymbol{x}_{(k)}$ have had the event at $T_{(k)}$. The risk set at t is denoted by $\mathcal{R}(t) = \{l | Y_l(t) = 1\}$, where $Y_l(t)$ is 1 if the lth individual is at risk "just before" time t and 0 otherwise.

Then, the conditional probability of observing an event for individual i at time t, given that an event is observed at that time for an individual in the riskset at time t is expressed as

$$\pi(i|t) = \frac{\alpha(t|\boldsymbol{x}_i)}{\sum_{l \in R(t)} \alpha(t|\boldsymbol{x}_l)}$$

$$= \frac{exp(\boldsymbol{x}_i\boldsymbol{\beta})}{\sum_{l\in R(t)} exp(\boldsymbol{x}_l\boldsymbol{\beta})}. \tag{2.2.10}$$

The partial likelihood for $\boldsymbol{\beta}$ is defined by multiplying the conditional probabilities over all observed event times,

$$PL(\boldsymbol{\beta}) = \prod_{k=1}^{K} \frac{\exp\big(\boldsymbol{x}_{(k)}\boldsymbol{\beta}\big)}{\sum_{l\in\mathcal{R}(T_{(k)})} \exp(\boldsymbol{x}_l\boldsymbol{\beta})}. \tag{2.2.11}$$

The partial log-likelihood becomes

$$pl(\boldsymbol{\beta}) = \sum_{k=1}^{K} \left\{ \boldsymbol{x}_{(k)}\boldsymbol{\beta} - \log \sum_{l\in\mathcal{R}(T_{(k)})} \exp(\boldsymbol{x}_l\boldsymbol{\beta}) \right\}. \tag{2.2.12}$$

Or equivalently, by setting $\mathcal{R}_{(i)} = \mathcal{R}(\widetilde{T}_i)$, where $\widetilde{T}_i$ is the observed time for individual i,

$$pl(\boldsymbol{\beta}) = \sum_{i=1}^{n} d_i \left\{ \boldsymbol{x}_i\boldsymbol{\beta} - \log \sum_{l\in\mathcal{R}_{(i)}} \exp(\boldsymbol{x}_l\boldsymbol{\beta}) \right\}, \tag{2.2.13}$$

where d_i denotes the censoring indicator; $d_i = 1$ if we observe the true survival time, and 0 if the observation is censored. Via maximizing the partial log-likelihood, $\boldsymbol{\beta}$ can be estimated without considering the baseline hazard.

2.2.4 Boosting for Cox regression model

Maximizing the likelihood is equivalent to minimizing the loss function defined as the negative likelihood. To perform Cox regression, gradient boosting sets the negative partial log-likelihood as the loss function, and seek the estimation of the coefficients $\boldsymbol{\beta}$ by iteratively updating the values by the base learners (Ridgeway, 1999). Consider the survival data,

$$(\widetilde{T}_i, d_i, \boldsymbol{x}_i) \qquad \text{for} \quad i = 1, \ldots, n,$$

where $\widetilde{T}_i$ is the observed survival time, d_i is the censoring indicator, which has value 1 if the individual is not censored, and 0 otherwise. $\boldsymbol{x}_i$ is the p-dimensional predictor vector for the ith individual. From equation (2.2.13), the negative gradient vector $\boldsymbol{u}_i$ of the negative partial log-likelihood is computed as

$$\begin{aligned} \boldsymbol{u}_i &= \frac{\partial}{\partial \boldsymbol{x}_i\boldsymbol{\beta}} \sum_{v=1}^{n} d_v \left\{ \boldsymbol{x}_v\boldsymbol{\beta} - \log \sum_{l\in\mathcal{R}_{(v)}} \exp(\boldsymbol{x}_l\boldsymbol{\beta}) \right\} \\ &= d_i - \sum_{v=1}^{n} d_v I(i \in \mathcal{R}_{(v)}) \frac{\exp\{\boldsymbol{x}_i\boldsymbol{\beta}\}}{\sum\limits_{l\in\mathcal{R}_{(v)}} \exp\{\boldsymbol{x}_l\boldsymbol{\beta}\}}, \end{aligned} \tag{2.2.14}$$

where $v = 1, \ldots, n$ are indices of the observations as well as $i = 1, \ldots, n$.

By computing the negative gradient vector $\boldsymbol{u}_i$ as above, function `glmboost` in the R package mboost performs componentwise gradient boosting for survival

times, where pre-built function `CoxPH` are set for the option `family`. The base learners of `glmboost` are least-squares estimators. The overall procedure of glmboost for Cox regression is described in Algorithm 2.2.1. For more details on `glmboost` for the Cox regression, we refer to De Bin (2016) and a comprehensive tutorial of R package mboost is given in Hofner et al. (2014).

Algorithm 2.2.1 glmboost for Cox regression

1. Set iteration counter $m = 0$. Initialize the estimate $\hat{\boldsymbol{\beta}}^{[0]} = (\hat{\beta}_1^{[0]}, \ldots, \hat{\beta}_p^{[0]}) = (0, \ldots, 0)$.
2. For $m = 1, \ldots, m_{stop}$,
   1) Compute the gradient vector of the partial log-likelihood evaluated in the previous iteration,

$$\boldsymbol{u}_i = d_i - \sum_{v=1}^{n} d_v I(i \in \mathcal{R}_{(v)}) \frac{\exp\{\boldsymbol{x}_i \boldsymbol{\beta}\}}{\sum\limits_{l \in \mathcal{R}_{(v)}} \exp\{\boldsymbol{x}_l \boldsymbol{\beta}\}} \Bigg|_{\boldsymbol{\beta} = \hat{\boldsymbol{\beta}}^{[m-1]}} .$$

   2) By applying the least-squares estimator to the negative gradient vector $\boldsymbol{u}^{[m]} = (u_1^{[m]}, \ldots, u_n^{[m]})$ separately for $j = 1, \cdots, p$, obtain $\hat{\gamma}_j^{[m]}$,

$$\hat{\gamma_j}^{[m]} = \sum_{i=1}^{n} x_{ij} u_i / \sum_{i=1}^{n} {x_{ij}}^2,$$

   where x_{ij} is the jth component for the ith observation.

   3) Select the best component j^* such that

$$j^* = \underset{j}{\operatorname{argmin}} \sum_{i=1}^{n} (u_i^{[m]} - \hat{\gamma_j}^{[m]} x_{ij})^2.$$

   4) Update the estimates,

$$\hat{\beta_j}^{[m]} = \begin{cases} \hat{\beta}_j^{[m-1]} + \nu \hat{\gamma_j}^{[m]} & \text{if } j = j^* \\ \hat{\beta}_j^{[m-1]} & \text{otherwise,} \end{cases}$$

   where $0 < \nu < 1$ is the step-length factor.

2.3 Lasso

Lasso is a regularized linear regression method, which was introduced by Tibshirani (1996). This method shrinks the regression coefficients by penalizing their size measured in the L_1 norm. Let $y_i \in \mathbb{R}$ denote the response variable, $x_{ij} \in \mathbb{R}$ be the jth predictor for the ith observation and β_j denote the coefficient of the jth predictor. The lasso regression estimates the coefficients by minimizing a penalized residual sum of squares,

$$\hat{\boldsymbol{\beta}} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}, \tag{2.3.1}$$

where $\sum_{j=1}^{p} |\beta_j|$ is called L_1 penalty term and $\lambda \geq 0$ is a complexity parameter. As the value of λ increases, the amount of shrinkage increases and the coefficients shrink toward zero. If λ is 0, then the Lasso estimate is equivalent to the ordinary least-squares estimate, which tends to have low bias but high variance. The purpose of shrinkage is to control the complexity of the regression model, avoiding high variance at the expense of a slightly larger bias to enhance the overall prediction accuracy (Hastie et al., 2009). To find the optimal value of the complexity parameter λ, cross-validation (CV) is often used.

Equivalently, the Lasso estimates can be rewritten as

$$\begin{aligned} \hat{\boldsymbol{\beta}} = & \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 \right\} \\ & \text{subject to} \quad \sum_{j=1}^{p} |\beta_j| \leq t, \end{aligned} \tag{2.3.2}$$

where the size constraint on the coefficients is explicit. Due to the nature of this constraint, when t is sufficiently small, some of the coefficient estimates will be exactly zero, i.e., some predictors are eliminated from the model. Therefore, Lasso does a type of automated variable selection during the model estimation. It is a desirable characteristic in terms of interpretability, especially in the high-dimensional setting. Among a huge number of possible predictors, Lasso regression provides a smaller subset of predictors that exhibit the strongest effects, which allow the model easy to interpret.

A problem of Lasso appears when there are strong correlations among subsets of the predictors. Hastie et al. (2009) presented that a regularization path of Lasso in this situation, which displays the transition of the coefficients against the strength of the regulation, fluctuate widely. It shows that the coefficients of the predictors selected by lasso are not stable. Lasso tends to suffer from the multi-collinearity problem.

2.4 Priority-Lasso

2.4.1 Principles

Priority-Lasso is a practical analysis strategy to build a regression model based on Lasso, which uses a hierarchical approach to process multi-omics data (Klau et al., 2018). In recent years, there has been a lot of interest in incorporating high-dimensional omics data into predictive models, not only using one type of biomarkers, but also dealing with different types of markers, such as clinical data, gene expression data, methylation data, etc. Priority-Lasso has been developed to handle several groups of different types of variables including high-throughput molecular data. This makes a difference from standard Lasso, which does not consider such a group structure in data, i.e., Lasso does not distinguish different groups of data, while priority-Lasso does.

An important characteristic of priority-Lasso is that it favors models consisting of a small number of input variables selected from specific sets of "favorite" variables. It is often assumed that medical professionals have prior knowledge of the disease under study and they are aware of some variables that are expected to yield high prediction accuracy. Furthermore, variables that are easier and cheaper to collect, such as age and common clinical variables, may be preferred over those collected with newer technologies, even if they lead to (slightly) lower prediction accuracy.

In order to incorporate prior knowledge and/or practical constraints to the prediction model, we group the predictors into several blocks and assign priorities to each of the blocks. The principle of priority-Lasso is to process data in the hierarchical structure according to the priority order.

2.4.2 Algorithm

Here, we assume continuous input variables and continuous responses for simplicity. Also assume that each variable is centered to have mean zero. Let $\boldsymbol{x}_i \in \mathbb{R}^p$ denote the p-dimensional predictor vector and $y_i \in \mathbb{R}$ denote the response for the ith individual, $i = 1, \ldots, n$. Before running priority-Lasso, we need to specify the block structure of the data, i.e., to group the predictors into several blocks so that each predictor belongs to only one block. Assume that we divide the predictors into B blocks and the bth block has p_b predictors for $b = 1, \ldots, B$, where $p = \sum_{b=1}^{B} p_b$. The predictors from block b for the ith individual are denoted as $\boldsymbol{x}_i^{(b)} = (x_{i1}^{(b)}, \ldots, x_{ip_b}^{(b)})$. Furthermore,we set the priority order among the blocks. Let $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_B)$ denote the permutation of $(1, \ldots, B)$ that indicates the priority order of blocks. π_1 is the index of the block with the highest priority, while π_B is that with the lowest priority.

Then, the prediction model is fitted through a hierarchical approach. Let $\beta_j^{(b)}$ denote the Lasso regression coefficient for the jth predictor from block b for $j = 1, \ldots, p_b$. First, a Lasso regression model is fitted to block π_1, the block with the highest priority, i.e., we estimate $\hat{\boldsymbol{\beta}}^{(\pi_1)} = \hat{\beta}_1^{(\pi_1)}, \ldots, \hat{\beta}_{p_{\pi_1}}^{(\pi_1)}$, the Lasso coefficients for the predictors in block π_1, that minimize

$$\sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{p_{\pi_1}} x_{ij}^{(\pi_1)} \beta_j^{(\pi_1)} \right)^2 + \lambda^{(\pi_1)} \sum_{j=1}^{p_{\pi_1}} \left| \beta_j^{(\pi_1)} \right|. \tag{2.4.1}$$

We compute the fitted linear predictor, called also *linear score*, in the first step,

$$\hat{\eta}_{1,i}(\boldsymbol{\pi}) = \hat{\beta}_1^{(\pi_1)} x_{i1}^{(\pi_1)} + \ldots + \hat{\beta}_{p_{\pi_1}}^{(\pi_1)} x_{ip_{\pi_1}}^{(\pi_1)}. \tag{2.4.2}$$

Next, Lasso is fitted to the block with the second highest priority, using the fitted linear predictor from the first step as an offset. We estimate $\hat{\beta}_1^{(\pi_2)}, \ldots, \hat{\beta}_{p_{\pi_2}}^{(\pi_2)}$, the Lasso coefficients for the predictors in block π_2, that minimize

$$\sum_{i=1}^{n} \left(y_i - \hat{\eta}_{1,i}(\boldsymbol{\pi}) - \sum_{j=1}^{p_{\pi_2}} x_{ij}^{(\pi_2)} \beta_j^{(\pi_2)} \right)^2 + \lambda^{(\pi_2)} \sum_{j=1}^{p_{\pi_2}} \left| \beta_j^{(\pi_2)} \right|. \tag{2.4.3}$$

The linear predictor in the second step is computed as

$$\hat{\eta}_{2,i}(\boldsymbol{\pi}) = \hat{\eta}_{1,i}(\boldsymbol{\pi}) + \hat{\beta}_1^{(\pi_2)} x_{i1}^{(\pi_2)} + \ldots + \hat{\beta}_{p_{\pi_2}}^{(\pi_2)} x_{ip_{\pi_2}}^{(\pi_2)}. \tag{2.4.4}$$

Then similarly, a Lasso model is fitted to the block with the third highest priority, using the linear score from the second step as offset. All remaining blocks are treated in the same manner. When fitting is done until the block π_B, we obtain the set of the coefficient estimates $\hat{\boldsymbol{\beta}}^{(\pi_b)}$ for $b = 1, \ldots, B$, and the final fitted linear predictor becomes

$$\hat{f}_{\mathrm{PL}} = \hat{\eta}_{B,i}(\boldsymbol{\pi}) = \sum_{b=1}^{B} \sum_{j=1}^{p_{\pi_b}} \hat{\beta}_j^{(\pi_b)} x_{ij}^{(\pi_b)}. \tag{2.4.5}$$

Due to the hierarchical structure, the predictors from the first block are used to explain the largest possible part of the variability in the responses, and the predictors from the subsequent blocks enter the model only if they explain variability that could not be explained by the blocks with higher priorities. Thus, priority-Lasso builds models where predictors in blocks with higher priorities play a more important role.

2.4.3 Cross-validated offsets

As discussed previously, priority-Lasso processes data by setting the priority order among blocks, which allows the model to incorporate prior knowledge. However, this hierarchical structure may affect prediction accuracy negatively because there is possibility to underestimate the influences of predictors in blocks with lower priorities and to eliminate useful predictors in these blocks.

This problem is due to the fact that the linear score in each block (the offset for the next block), $\hat{\eta}_{b,i}(\boldsymbol{\pi})$, tends to be over-optimistic with respect to the effects of the predictors in the bth block on the response y_i (Klau et al., 2018). This problem is similar to the well-known overoptimism regarding prediction error, i.e, if we compute the prediction error on the training dataset where we built the prediction model, it is presumed that the error shows a smaller value than the generalization error. Similarly, the overoptimism problem of the linear scores arises from the fact that y_i were included in the data used to estimate the coefficients $\hat{\beta}_1^{(\pi_b)}, \ldots, \hat{\beta}_{p_{\pi_b}}^{(\pi_b)}$, which are then used to compute the linear score $\hat{\eta}_{b,i}(\boldsymbol{\pi})$. This overly optimistic estimate $\hat{\eta}_{b,i}(\boldsymbol{\pi})$ is presumed to include some of

variability in y_i that cannot actually be explained by this block, but might be explained by the following blocks. Using this estimate as offset deprives the opportunity for the following blocks to explain this part of variability in y_i.

To solve this problem, the cross-validated offsets, $\hat{\eta}_{b,i}(\boldsymbol{\pi})_{CV}$, which are estimated using cross-validation (CV) were devised. Consider that we have dataset S and use K-fold CV to compute offsets. The cross-validated offsets are obtained as follows.

1. Split the dataset S randomly into K parts, $S_1, \ldots, S_K$.

2. For $k = 1, \ldots, K$, estimate Lasso coefficients $\hat{\beta}^{(\pi_b)}_{S\setminus S_k,1}, \ldots, \hat{\beta}^{(\pi_b)}_{S\setminus S_k,p_{\pi_b}}$ by using the training data $S \setminus S_k$ for $i \in S_k$, where $S \setminus S_k$ is the set obtained by removing the the observations of S_k from S.

3. Compute the cross-validated offsets,

$$\hat{\eta}_{b,i}(\boldsymbol{\pi})_{CV} = \hat{\eta}_{b-1,i}(\boldsymbol{\pi})_{CV} + \hat{\beta}^{(\pi_b)}_{S\setminus S_k,1} x^{(\pi_b)}_{i1} + \cdots + \hat{\beta}^{(\pi_b)}_{S\setminus S_k,p_{\pi_1}} x^{(\pi_b)}_{ip_{\pi_b}}, \quad (2.4.6)$$

   where $\hat{\eta}_{0,i}(\boldsymbol{\pi})_{CV} = 0$.

Priority-Lasso is implemented in the function `prioritylasso` from R package of the same name. This function has the option `cvoffset`, where we select whether to use the cross-validated offsets or the standard offsets estimated without cross-validation, which are shown in the algorithm in the previous section,

$$\hat{\eta}_{b,i}(\boldsymbol{\pi}) = \hat{\eta}_{b-1,i}(\boldsymbol{\pi}) + \hat{\beta}^{(\pi_b)}_{1} x^{(\pi_b)}_{i1} + \ldots + \hat{\beta}^{(\pi_b)}_{p_{\pi_b}} x^{(\pi_b)}_{ip_{\pi_b}}, \quad (2.4.7)$$

where $\hat{\eta}_{0,i}(\boldsymbol{\pi}) = 0$

The version with cross-validated offsets helps to avoid underestimating the influence of blocks with lower priorities. On the other hand, this version is more computationally expensive, and therefore it may not be easily applicable to all cases. Also, the version with the standard offsets may be a more practical choice when there are groups where one would preferably like to reduce the number of predictors, e.g., blocks consisting predictors that are costly to collect (Klau et al., 2018).

CHAPTER 3

Novel approaches

3.1 Priority boosting

3.1.1 Principles

Priority boosting is a practical regression method for high-dimensional data consisting of multiple groups of variables, such as multi-omics data. It was developed based on the same strategy as priority-Lasso. Namely, we define the priority order among the groups and process the data according to the order. The aim of the strategy is to build a model that (preferably) includes predictors from particular groups consisting of "favorite" variables. Thus, this approach enables the model to incorporate user's preference.

Priority boosting builds a prediction model based on the componentwise boosting. Some characteristics of the componentwise boosting are passed on to priority boosting. One key characteristic is that it performs automated variable selection during the model estimation. Another property is that the resulting model of priority boosting has the same form of the additive predictor in the generalized additive model (GAM),

$$\beta_0 + h_1(x_1) + h_2(x_2) + \cdots + h_p(x_p), \tag{3.1.1}$$

which is interpretable as discussed in Section 2.1.4.

Here, we provide a succinct overview of the priority boosting algorithm, while a formal description can be found in Section 3.1.2. The structure of priority boosting is quite similar to priority-Lasso. Firstly, we define the group structure (block structure) of the predictors, i.e., divide predictors into multiple blocks so that each predictor belongs to only one block. Then, we specify the priority order of these blocks. For example, the block with common clinical variables that are easier to collect might be set higher priority than the block consisting of variables measured with expensive technologies.

According to the priority order, the prediction model is fitted in a stepwise manner. Fitting starts with the block with the highest priority and proceeds until the block with the lowest priority, where the prediction model, $\hat{f}_{PB}$, is obtained. While in priority-Lasso, fitting with each block is performed by applying Lasso regression, priority boosting applies the componentwise gradient boosting.

To apply the componentwise gradient boosting to each block, we should define the base learners (base procedures). One choice is to use least-squares

estimators. Then, where the conditional distribution of the response is assumed to follow an exponential family distribution, this componentwise gradient boosting fits a *generalized linear model* (GLM), which has the form

$$g(E(Y|\boldsymbol{x})) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p, \tag{3.1.2}$$

where $\eta(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$ is called the *linear predictor* and g is a link function that is strictly increasing and differentiable. Note that GLM, which only models linear effects of variables, is a model class included in GAM. The componentwise gradient boosting that uses least-squares estimators as base learners is implemented by function `glmboost` in R package mboost (Hothorn et al., 2021). If we want to model nonlinear effects of variables, then the function `gamboost` in mboost can be used, which implements the componentwise gradient boosting that fits GAM. However, in the following, we focus on priority boosting that uses `glmboost`, i.e., the componentwise gradient boosting that uses least-squares estimators as the base learners. Note that as discussed in Section 2.1.4, `glmboost` and `gamboost` can be used even though the distribution is not assumed to follow an exponential family distribution. The resulting models have the same form as the linear predictor in GLM and the additive predictor in GAM respectively.

In addition to setting the base learners, to perform the componentwise gradient boosting, we need to define the loss function and compute its negative gradient. For continuous response Y and continuous input variables $\boldsymbol{X}$, if we assume that $Y|\boldsymbol{X} = \boldsymbol{x}$ is normally distributed, then we define the L_2 loss as the loss function, which is equivalent to the negative Gaussian log-likelihood. For the cases where responses are survival times, our aim is to estimate the linear predictor $\eta = \boldsymbol{x}\boldsymbol{\beta}$ in the hazard rate (the relative risk function) $\alpha(t|\boldsymbol{x})$ in the Cox regression model under the proportional hazards assumption. As expressed in equation (2.2.8), the hazard rate in the Cox model takes the form,

$$\alpha(t|\boldsymbol{x}) = \alpha_0(t) \exp\{\boldsymbol{x}\boldsymbol{\beta}\}.$$

We can perform the Cox regression by setting the negative partial log-likelihood as the loss function, which was discussed in Section 2.2.4 The algorithm of `glmboost` for Cox model is shown in Algorithm 2.2.1.

3.1.2 Algorithm

Here, we assume continuous input variables $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{ip})$ and continuous responses y_i for $i = 1, \ldots, n$. Also assume that variables are centered for simplicity. We refer the componentwise gradient boosting with least-squares base learners and L_2 loss as the L_2 glmboost shortly. Consider the same notation as those in Section 2.4; Suppose that we have B blocks and $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_B)$ denotes the permutation of $(1, \ldots, B)$ that indicates the priority order of blocks. π_1 is the index of the block with highest priority, while π_B is that with the lowest priority. The bth block has p_b predictors for $b = 1, \ldots, B$. The predictors from block b for the ith individual are denoted as $\boldsymbol{x}_i^{(b)} = (x_{i1}^{(b)}, \ldots, x_{ip_b}^{(b)})$. Let $\beta_j^{(b)}$ be the coefficient of the jth predictor from block b for $j = 1, \ldots, p_b$.

First, we apply the L_2 glmboost to block π_1, the block with the highest priority, and obtain $\hat{\beta}_1^{(\pi_1)}, \ldots, \hat{\beta}_{p_{\pi_1}}^{(\pi_1)}$, the coefficient estimates of the variables

from block π_1. Then, we compute the fitted linear predictor from the first step,

$$\hat{\eta}_{1,i}(\boldsymbol{\pi}) = \hat{\beta}_1^{(\pi_1)} x_{i1}^{(\pi_1)} + \ldots + \hat{\beta}_{p_{\pi_1}}^{(\pi_1)} x_{ip_{\pi_1}}^{(\pi_1)}. \tag{3.1.3}$$

Next, we apply the L_2 glmboost to block π_2, using the fitted linear predictor from the first step as an offset, and obtain $\hat{\beta}_1^{(\pi_2)}, \ldots, \hat{\beta}_{p_{\pi_2}}^{(\pi_2)}$, the coefficient estimates of the variables from block π_2. The fitted linear predictor from the second step is computed as

$$\hat{\eta}_{2,i}(\boldsymbol{\pi}) = \hat{\eta}_{1,i}(\boldsymbol{\pi}) + \hat{\beta}_1^{(\pi_2)} x_{i1}^{(\pi_2)} + \ldots + \hat{\beta}_{p_{\pi_2}}^{(\pi_2)} x_{ip_{\pi_2}}^{(\pi_2)}. \tag{3.1.4}$$

Then, similarly we apply the L_2 glmboost to block π_3, using the fitted linear predictor from the second step as an offset, estimating $\hat{\beta}_1^{(\pi_3)}, \ldots, \hat{\beta}_{p_{\pi_3}}^{(\pi_3)}$, the coefficient estimates of the variables from block π_3. Priority boosting proceeds in the same way for the remaining block until the block with the last priority, estimating the coefficients of the predictors from the corresponding block. The final fitted linear predictor is expressed as

$$\hat{f}_{\mathrm{PB}}(\boldsymbol{x}_i) = \hat{\eta}_{B,i}(\boldsymbol{\pi}) = \sum_{b=1}^{B} \sum_{j=1}^{p_{\pi_b}} \hat{\beta}_j^{(\pi_b)} x_{ij}^{(\pi_b)}. \tag{3.1.5}$$

A schematic overview is shown in Algorithm 3.1.1.

Extension to time-to-event responses can be done by replacing the L_2 glmboost with the `glmboost` for Cox model, which is mentioned above. $\hat{f}_{\mathrm{PB}}$ is the estimate of the linear predictor in the hazard rate.

Note that the hyper parameters of priority boosting are the parameters of `glmboost` that we apply to each block; the step-length factor ν and the stopping number of iteration m_{stop}. These parameters control shrinkage of the estimate and also the model sparsity, which we discussed in section 2.1.5. A comprehensive tutorial of `glmboost` is given in Hofner et al. (2014). We implemented priority boosting for continuous responses and time-to-event responses in R function `prioritybоost`. The code of `prioritybоost` is shown in Appendix A.

Algorithm 3.1.1 Priority boosting

1. Specify the block structure of the predictors, where each predictor belongs to only one of the B blocks.
   Specify the priority order of these blocks $\boldsymbol{\pi} = (\pi_1, ..., \pi_B)$.
2. For block π_1,
   1) Estimate $\hat{\beta}_1^{(\pi_1)}, \ldots, \hat{\beta}_{p_{\pi_1}}^{(\pi_1)}$, the coefficients of the predictors from block π_1, by running `glmboost` for block π_1.
   2) Compute the fitted linear predictor,

$$\hat{\eta}_{1,i}(\boldsymbol{\pi}) = \hat{\beta}_1^{(\pi_1)} x_{i1}^{(\pi_1)} + \ldots + \hat{\beta}_{p_{\pi_1}}^{(\pi_1)} x_{ip_{\pi_1}}^{(\pi_1)}.$$

3. From block π_2 to block π_B,
   1) Estimate $\hat{\beta}_1^{(\pi_b)}, \ldots, \hat{\beta}_{p_{\pi_b}}^{(\pi_b)}$, the coefficient of the predictors π_b, by running `glmboost` for block π_b with $\hat{\eta}_{b-1,i}(\boldsymbol{\pi})$, the linear predictor fitted in the previous step, as an offset.
   2) Compute the fitted linear predictor,

$$\hat{\eta}_{b,i}(\boldsymbol{\pi}) = \hat{\eta}_{b-1,i}(\boldsymbol{\pi}) + \hat{\beta}_1^{(\pi_b)} x_{i1}^{(\pi_b)} + \ldots + \hat{\beta}_{p_{\pi_b}}^{(\pi_b)} x_{ip_{\pi_b}}^{(\pi_b)}.$$

4. Using the coefficient estimates obtained from each block, $\hat{\beta}_1^{(\pi_b)}, \ldots, \hat{\beta}_{p_{\pi_b}}^{(\pi_b)}$, the final fitted linear predictor is computed as

$$\hat{f}_{PB}(\boldsymbol{x}_i) = \hat{\eta}_{B,i}(\boldsymbol{\pi}) = \sum_{b=1}^{B} \sum_{j=1}^{p_{\pi_b}} \hat{\beta}_j^{(\pi_b)} x_{ij}^{(\pi_b)}.$$

3.2 Lasso-based block boosting

3.2.1 Principles

Lasso-based block boosting (hereinafter called LBboost) is a regression method for high-dimensional data consisting of multiple groups. LBboost uses the block structure in data, that is, distinguishes the different groups in predictors. However, unlike priority-Lasso and priority boosting, it does not assign priorities to each of the groups.

As discussed in Section 2.4 and Section 3.1, defining the priority order among the groups allows us to build a model taking prior knowledge and/or practical constraints into account. In that sense, the resulting models of two priority methods are not derived completely data-driven. These methods favor to select predictors from specific sets consisting of "favorite" variables, and it might lead to lower predictive accuracy. We assume that LBboost will be an alternative method in the cases where it is wise to distinguish the different groups in data, but we have no basis for setting an appropriate priority order among the groups, or we hope to perform estimation purely data-driven.

Here, we introduce some key features of this method. First, LBboost inherits some useful features from Lasso regression. As with other boosting methods, LBboost estimates the regression function f via iteratively updating its value by applying the base learners at each boosting iteration. The base learners (base procedures) of LBboost are L_1 penalized least-squares estimator. Namely, a Lasso regression model is obtained as the fitted base learner at each iteration. Then, the resulting model of LBboost $\hat{f}_{\mathrm{LB}}(\boldsymbol{x}_i)$ is the linear combination of the Lasso models, and it is expressed as

$$\hat{f}_{\mathrm{LB}}(\boldsymbol{x}_i) = \hat{\beta}_0^{[m_{\mathrm{stop}}]} + \hat{\beta}_1^{[m_{\mathrm{stop}}]} x_{i1} + \ldots + \hat{\beta}_p^{[m_{\mathrm{stop}}]} x_{ip}, \tag{3.2.1}$$

which is the same form as the linear predictor in GLM in equation (3.1.2),

$$g(E(Y|\boldsymbol{x})) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p.$$

We can see that LBboost provides an interpletable model. LBboost incorporates automated variable selection, which is also a feature inherited from Lasso regression.

Another key characteristic is that LBboost processes data by what we call the *subset-updating approach*. Assume that we divide predictors into B disjoint subsets (blocks). At each iteration, fitting is performed separately for each block. Namely, every fitting is done only using the predictors in the corresponding block. Consequently, we obtain B fitted base learners and choose the base learner that provides the "best" update. The best update is selected, for example, in terms of giving the smallest empirical risk.

To get a big picture of the subset-updating approach, it may be useful to consider again the difference between the updating manner in FGD (Algorithm 2.1.2) in Section 2.1.3 and that in componentwise gradient boosting (Algorithm 2.1.3) in Section 2.1.4. In each boosting iteration, FGD updates the coefficients of all the predictors, whereas the componentwise boosting updates the coefficient of only one predictor at a time. The updating manner of LBboost lies between FGD and componentwise boosting; It updates the coefficient from only one

subset of predictors at each iteration. If the number of subsets B is 1, then LBboost will update all the coefficients of predictors, and if B is equal to the number of predictors, then it updates the coefficient of only one predictor, which is the exactly componentwise approach.

Note that here, we used FGD and componentwise gradient boosting for analogies of updating manner, but we will not introduce LBboost as gradient boosting. Rather, LBboost is defined as a boosting with offset-based procedure, such as a well-known boosting algorithm, likelihood-based boosting. At each iteration, we compute the function estimate (the fitted linear predictor) $\hat{f}$, and the value is included as an offset in the fit at the next iteration. We provide the algorithm of LBboost formally in section 3.2.2.

3.2.2 Algorithm

In the following, we assume continuous input variables $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{ip})$ and continuous responses y_i for $i = 1, \ldots, n$. Also assume that variables are centered for simplicity. The first step of LBboost is to specify the block structure of the predictors, namely, to divide predictors into B blocks so that each predictor belongs to only one block. In the case of multi-omics analysis, we may divide predictors by specific data types, such as block for clinical variables or block for gene expression variables. Let b denote the index of the blocks, $b = 1, \ldots, B$, and assume that block b consists of p_b predictors. Let $\boldsymbol{x}_i^{(b)}$ be the p_b-dimensional vector consisting of the predictors in block b for the ith observation, and $x_{ij}^{(b)}$ denotes the jth component of $\boldsymbol{x}_i^{(b)}$.

We set the initial guess of the coefficient estimates, $\hat{\boldsymbol{\beta}}^{[0]} = (\hat{\beta}_1^{[0]}, \ldots, \hat{\beta}_p^{[0]}) = (0, \ldots, 0)$ and initialize the fitted linear predictor, $f^{[0]}(\boldsymbol{x}_i) = 0$ for all i. At the first iteration, a Lasso regression model is fitted to each block b. Namely, we estimate coefficients $\hat{\boldsymbol{\gamma}}^{(b)[1]} = (\hat{\gamma}_1^{(b)[1]}, \ldots, \hat{\gamma}_{p_b}^{(b)[1]})$, that minimize

$$\sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{p_b} x_{ij}^{(b)} \hat{\gamma}_j^{(b)[1]} \right)^2 + \lambda^{(b)[1]} \sum_{j=1}^{p_b} \left| \hat{\gamma}_j^{(b)[1]} \right|. \tag{3.2.2}$$

Among B blocks, we select the block b^* that gives the best update. For example, we may choose the block that minimizes the empirical risk the most,

$$\begin{aligned} b^* &= \underset{b \in \{1, \ldots, B\}}{\operatorname{argmin}} \sum_{i=1}^{n} L \left(y_i, \sum_{j=1}^{p_b} x_{ij}^{(b)} \hat{\gamma}_j^{(b)[1]} \right) \\ &= \underset{b \in \{1, \ldots, B\}}{\operatorname{argmin}} \sum_{i=1}^{n} L \left(y_i, \boldsymbol{x}_i^{(b)} \hat{\boldsymbol{\gamma}}^{(b)[1]} \right). \end{aligned} \tag{3.2.3}$$

One common choice of the loss function in equation (3.2.3) is L_2 loss.

As in gradient boosting, to obtain small updates, we update the coefficient estimates with step-length factor $0 < \nu < 1$.

$$\hat{\beta}_j^{(b)[1]} = \begin{cases} \nu \hat{\gamma}_j^{(b)[1]} & \text{if } b = b^* \\ 0 & \text{otherwise,} \end{cases} \tag{3.2.4}$$

where $\hat{\beta}_j^{(b)[1]}$ denotes the coefficient estimate for the jth predictor in block b for $j = 1, \ldots, p_b$ at the first iteration. Let $\hat{\boldsymbol{\beta}}^{(b)[1]}$ denote the p_b dimensional vector consisting of $\hat{\beta}_j^{(b)[1]}$. The fitted linear predictor at the first iteration is

$$\begin{aligned}\hat{f}^{[1]}(\boldsymbol{x}_i) &= \hat{f}^{[0]}(\boldsymbol{x}_i) + \nu \boldsymbol{x}_i^{(b^*)} \hat{\boldsymbol{\gamma}}^{(b^*)[1]} \\ &= \sum_{b=1}^{B} \boldsymbol{x}_i^{(b)} \hat{\boldsymbol{\beta}}^{(b)[1]} = \boldsymbol{x}_i \hat{\boldsymbol{\beta}}^{[1]}. \end{aligned} \tag{3.2.5}$$

In the fit at the second iteration, the fitted linear predictor (also called linear score), $\hat{f}^{[1]}$, is included as an offset. Namely, we estimate coefficients $\hat{\boldsymbol{\gamma}}^{(b)[2]} = (\hat{\gamma}_1^{(b)[2]}, \ldots, \hat{\gamma}_{p_b}^{(b)[2]})$ that minimize

$$\sum_{i=1}^{n} \left(y_i - \hat{f}^{[1]}(\boldsymbol{x}_i) - \sum_{j=1}^{p_b} x_{ij}^{(b)} \hat{\gamma}_j^{(b)[2]} \right)^2 + \lambda^{(b)[2]} \sum_{j=1}^{p_b} \left| \hat{\gamma}_j^{(b)[2]} \right|. \tag{3.2.6}$$

By including $\hat{f}^{[1]}$ as an offset, a Lasso model will be fitted to the residuals from the first iteration. Then, we select the block b^* that gives the best update, e.g.,

$$\begin{aligned} b^* &= \underset{b \in \{1,\ldots,B\}}{\operatorname{argmin}} \sum_{i=1}^{n} L \left(y_i, \hat{f}^{[1]}(\boldsymbol{x}_i) + \sum_{j=1}^{p_b} x_{ij}^{(b)} \hat{\gamma}_j^{(b)[2]} \right) \\ &= \underset{b \in \{1,\ldots,B\}}{\operatorname{argmin}} \sum_{i=1}^{n} L \left\{ y_i, \hat{f}^{[1]}(\boldsymbol{x}_i) + \boldsymbol{x}_i^{(b)} \hat{\boldsymbol{\gamma}}^{(b)[2]} \right\}. \end{aligned} \tag{3.2.7}$$

The updating equation is

$$\hat{\beta}_j^{(b)[2]} = \begin{cases} \hat{\beta}_j^{(b)[1]} + \nu \hat{\gamma}_j^{(b)[2]} & \text{if } b = b^* \\ \hat{\beta}_j^{(b)[1]} & \text{otherwise.} \end{cases} \tag{3.2.8}$$

Compute the fitted linear predictor at the second iteration

$$\begin{aligned}\hat{f}^{[2]}(\boldsymbol{x}_i) &= \hat{f}^{[1]}(\boldsymbol{x}_i) + \nu \boldsymbol{x}_i^{(b^*)} \hat{\boldsymbol{\gamma}}^{(b^*)[2]} \\ &= \sum_{b=1}^{B} \boldsymbol{x}_i^{(b)} \hat{\boldsymbol{\beta}}^{(b)[2]} = \boldsymbol{x}_i \hat{\boldsymbol{\beta}}^{[2]}. \end{aligned} \tag{3.2.9}$$

Similarly, at the third iteration, fitting is performed including the fitted linear predictor at the second iteration, $\hat{f}^{[2]}$, as an offset. This procedure is repeated to reach the stopping iteration m_{stop}, where we obtain the final fitted linear predictor, $\hat{f}_{\text{LB}}(\boldsymbol{x}_i) = \hat{f}^{[m_{\text{stop}}]}(\boldsymbol{x}_i) = \boldsymbol{x}_i \hat{\boldsymbol{\beta}}^{[m_{\text{stop}}]}$. A schematic overview of LBboost is shown in Algorithm 3.2.1.

Extension to time-to-event responses can be done using a variant of Lasso for Cox regression model (Tibshirani, 1997). In this case, when selecting the best update at each iteration, we may set the negative partial likelihood as the loss function. The final model $\hat{f}_{\text{LB}}(\boldsymbol{x}_i) = \boldsymbol{x}_i \hat{\boldsymbol{\beta}}^{[m_{\text{stop}}]}$ estimates the linear predictor in the hazard rate.

Algorithm 3.2.1 Lasso-based block boosting

1. Specify the block structure of the predictors, where each predictor belongs to only one of the B blocks.
   Set iteration counter $m = 0$. Initialize the coefficient estimates; $\hat{\boldsymbol{\beta}}^{[0]} = (\hat{\beta}_1^{[0]}, \ldots, \hat{\beta}_p^{[0]}) = (0, \ldots, 0)$, and initialize the fitted linear predictor; $\hat{f}^{[0]}(\boldsymbol{x}_i) = \boldsymbol{x}_i \hat{\boldsymbol{\beta}}^{[0]} = 0$.
2. For $m = 1, \ldots, m_{\text{stop}}$,
   2) For $b = 1 \ldots, B$, apply Lasso regression to block b with the fitted linear predictor at the previous iteration as an offset, i.e., estimate coefficients $\hat{\boldsymbol{\gamma}}^{(b)[m]} = (\hat{\gamma}_1^{(b)[m]}, \ldots, \hat{\gamma}_{p_b}^{(b)[m]})$ that minimize

$$\sum_{i=1}^{n} \left(y_i - \hat{f}^{[m-1]}(\boldsymbol{x}_i) - \sum_{j=1}^{p_b} x_{ij}^{(b)} \hat{\gamma}_j^{(b)[m]} \right)^2 + \lambda^{(b)[m]} \sum_{j=1}^{p_b} \left| \hat{\gamma}_j^{(b)[m]} \right|,$$

   where $x_{ij}^{(b)}$ denotes the jth component of $\boldsymbol{x}_i^{(b)}$ that is the p_b-dimensional vector consisting of the predictors in block b for the ith observation.
   3) Select the block b^* that gives the best update, e.g., the block that minimizes the empirical risk the most.
   4) Update the estimates for the coefficients,

$$\hat{\beta}_j{}^{(b)[m]} = \begin{cases} \hat{\beta}_j{}^{(b)[m-1]} + \nu \hat{\gamma}_j{}^{(b)[m]} & \text{if } b = b^* \\ \hat{\beta}_j{}^{(b)[m-1]} & \text{otherwise,} \end{cases}$$

   where $\hat{\beta}_j{}^{(b)[m]}$ denotes the coefficient estimate for the j th predictor in block b for $j = 1, \ldots, p_b$ at iteration m, and $0 < \nu < 1$ is the step-length factor.
   4) Compute the fitted linear predictor,

$$\begin{aligned} \hat{f}^{[m]}(\boldsymbol{x}_i) &= \hat{f}^{[m-1]}(\boldsymbol{x}_i) + \nu \boldsymbol{x}_i^{(b^*)} \hat{\boldsymbol{\gamma}}^{(b^*)[m]} \\ &= \sum_{b=1}^{B} \boldsymbol{x}_i^{(b)} \hat{\boldsymbol{\beta}}^{(b)[m]} = \boldsymbol{x}_i \hat{\boldsymbol{\beta}}^{[m]}. \end{aligned}$$

3. The final fitted linear predictor is

$$\hat{f}_{\text{LB}}(\boldsymbol{x}_i) = \hat{f}^{[m_{\text{stop}}]}(\boldsymbol{x}_i) = \sum_{b=1}^{B} \boldsymbol{x}_i^{(b)} \hat{\boldsymbol{\beta}}^{(b)[m_{\text{stop}}]} = \boldsymbol{x}_i \hat{\boldsymbol{\beta}}^{[m_{\text{stop}}]}.$$

CHAPTER 4

Simulation study

In this chapter, we asses the performances of priority boosting and Lasso-based block boosting (LBboost) on simulation data. Our simulation study has two parts; Part I is for Gaussian responses and Part II is for time-to-event responses. In the part for Gaussian cases, we have simulation data as pairs $(y_i, \boldsymbol{x_i})$ for $i = 1, \ldots, n$, where $y_i \in \mathbb{R}$ is the response variable and $\boldsymbol{x}_i \in \mathbb{R}^p$ is the input vector for the ith observation, and n is the number of observations. We assume that y_i is expressed as

$$y_i = \boldsymbol{x}_i\boldsymbol{\beta} + \epsilon_i,$$

where $\boldsymbol{\beta}$ is the coefficient vector and ϵ_i is a random noise that follows a normal distribution. In the part for survival data, we generate the simulation data as

$$(\widetilde{T}_i, d_i, \boldsymbol{x}_i) \qquad \text{for} \quad i = 1, \ldots, n,$$

where $\widetilde{T}_i$ is the observed time and d_i is the censoring indicator, which has value 1 if the individual is not censored, and 0 otherwise. In the following sections, we explain the simulation study design including how to generate data, and evaluate the prediction abilities of the above two methods comparing three other methods, priority-Lasso, Lasso and gradient boosting.

4.1 Simulation design: data and scenarios

4.1.1 Scenarios

When dealing with multi-omics datasets, various correlations need to be considered. There may be correlations within clinical variables or within gene expression variables. There may also be overlap of predictive information among different types of data, such as between clinical covariates and omics covariates. To simulate a multi-omics datasets, we consider three settings in terms of correlation structure.

Setting 1. Independent variables: All variables are independent.

Setting 2. Dependencies within each group: There are dependencies between variables within a group, but no dependency between variables belonging to different groups.

Setting 3. Dependencies among groups: There are dependencies among groups, namely, there are dependencies between

variables, regardless of whether they are in the same group or in different groups.

Furthermore, we consider two settings regarding the number of the input variables. We assume that the predictors are grouped into four blocks. 1) LLLH: block 1 to block 3 consists of 10 variables and block 4 consists of 1000 variables, 2) LHHH: block 1 consists of 10 variables and block 2 to block 4 consists of 1000 variables. These settings mimic data consisting of low-dimensional clinical variables and low- or high-dimensional omics variables.

Then, we have the three settings regarding the correlation structure among variables and the two settings in terms of the dimension of predictors in each block, which summed up in six scenarios;

Scenario 1. LLLH, Independent variables
Scenario 2. LLLH, Dependencies within each group
Scenario 3. LLLH, Dependencies among groups
Scenario 4. LHHH, Independent variables
Scenario 5. LHHH, Dependencies within each group
Scenario 6. LHHH, Dependencies among groups

Following these six scenarios, we generated the predictors, which are common in the part for Gaussian cases and in the part for survival analysis.

4.1.2 Data generation

Generating predictors

To generate input variables in the simulation study, we define the variance-covariance matrix corresponding to each scenario, and then sample multivariate normally distributed random values following the correlation structure. We assume to generate $n \times p$ design matrix X, i.e, predictor matrix consisting of p variables for n individuals. Further, the p predictors are grouped into B blocks and block b consists of p_b predictors. $X^{(b)}$ denotes the design matrix consisting of the variables in block b.

In setting 1, since there is no correlation between the variables, the corresponding variance-covariance matrix Σ will be the $p \times p$ identity matrix,

$$\Sigma = \mathbb{1}_p. \tag{4.1.1}$$

Then, we sample the p-dimensional predictor vector for the ith observation, $\boldsymbol{x}_i$, from the multivariate normal distribution,

$$\boldsymbol{x}_i \sim N(\boldsymbol{\mu}, \Sigma), \tag{4.1.2}$$

where we set the mean vector $\boldsymbol{\mu}$ to the p-dimensional zero vector.

Sampling from a multivariate normal distribution is available by R package MASS. By running `mvrnorm(n, mu, sgm)`, where `mu` is the mean vector and `sgm` is the variance-covariance matrix, we obtain a design matirix X consisting of normally distributed variables. Alternatively, since here the covariance matrix is the identity matrix, `matrix(rnorm(n*p),n,p)` can be used.

In setting 2, we need to specify the correlation between variables within a group. Let denote the covariance matrix corresponding to block b by $\Sigma^{(b)}$ and the r, c component of $\Sigma^{(b)}$ by $\sigma_{rc}^{(b)}$. Note that $\Sigma^{(b)}$ is a $p_b \times p_b$ matrix. Here, we set $\sigma_{rc}^{(b)} = 0.9^{|r-c|}$ for all $b = 1, \ldots, B$, nemely,

$$\Sigma^{(b)} = \sigma_{rc}^{(b)} = 0.9^{|r-c|} = \begin{pmatrix} 1 & 0.9^1 & \ldots & 0.9^{|1-p_b|} \\ 0.9^1 & 1 & \ldots & 0.9^{|2-p_b|} \\ \vdots & \vdots & \ddots & \vdots \\ 0.9^{|p_b-1|} & 0.9^{|p_b-2|} & \ldots & 1 \end{pmatrix}. \quad (4.1.3)$$

Then, we sample $\boldsymbol{x}_i^{(b)}$, the p_b-dimensional normally distributed predictors in block b for the ith observation,

$$\boldsymbol{x}_i^{(b)} \sim N(\boldsymbol{\mu}^{(b)}, \Sigma^{(b)}), \quad (4.1.4)$$

setting the mean vector to the p_b-dimensional zero vector. Note that $\boldsymbol{x}_i^{(b)}$ is the ith row of $X^{(b)}$. Then, by combining $X^{(1)}, \ldots, X^{(B)}$, we obtain the predictor matrix X including all variables,

$$X = \left(\begin{array}{c|c|c|c} X^{(1)} & X^{(2)} & \ldots & X^{(B)} \end{array} \right). \quad (4.1.5)$$

The covariance matrix Σ in setting 3 is defined by $\sigma_{rc} = 0.9^{|r-c|}$, where σ_{rc} denotes the r, c component of Σ,

$$\Sigma = \sigma_{rc} = 0.9^{|r-c|} = \begin{pmatrix} 1 & 0.9^1 & \ldots & 0.9^{|1-p|} \\ 0.9^1 & 1 & \ldots & 0.9^{|2-p|} \\ \vdots & \vdots & \ddots & \vdots \\ 0.9^{|p-1|} & 0.9^{|p-2|} & \ldots & 1 \end{pmatrix}. \quad (4.1.6)$$

Note that Σ is a $p \times p$ matrix. Setting the mean vector $\boldsymbol{\mu}$ to the p-dimensional zero vector, we sample $\boldsymbol{x}_i$ following a multivariate normal distribution, $\boldsymbol{x}_i \sim N(\boldsymbol{\mu}, \Sigma)$.

Generating Gaussian responses

The response variables y_i for Gaussian cases were computed as

$$y_i = \boldsymbol{x}_i \boldsymbol{\beta} + \epsilon_i, \quad (4.1.7)$$

where $\boldsymbol{\beta}$ is the coefficient vector, and ϵ_i denotes Gaussian noise, which follows the standard normal distribution with the mean is 0 and the variance is 1, $\epsilon_i \sim N(0, 1)$. We set the coefficients as follows.

1) Scenario 1-3

Block 1 (10 variables): $\beta_1 = (0.9, 0.9, 0.9, 0.9, 0.9, 0, 0, 0, 0, 0)$
Block 2 (10 variables): $\beta_2 = (0.7, 0.7, 0.7, 0.7, 0.7, 0, 0, 0, 0, 0)$
Block 3 (10 variables): $\beta_3 = (0.3, 0.3, 0.3, 0.3, 0.3, 0, 0, 0, 0, 0)$
Block 4 (1000 variables): $\beta_4 = (0.1, 0.1, 0.1, 0.1, 0.1, 0, \ldots, 0)$

2) Scenario 4-6

Block 1 (10 variables): $\beta_1 = (0.9, 0.9, 0.9, 0.9, 0.9, 0, 0, 0, 0, 0)$
Block 2 (1000 variables): $\beta_2 = (0.7, 0.7, 0.7, 0.7, 0.7, 0, \ldots, 0)$

Block 3 (1000 variables): $\beta_3 = (0.3, 0.3, 0.3, 0.3, 0.3, 0, \ldots, 0)$
Block 4 (1000 variables): $\beta_4 = (0.1, 0.1, 0.1, 0.1, 0.1, 0, \ldots, 0)$

As can be seen above, the coefficients were set assuming that five predictors in each block have effect on the responses and predictors in Block 1 has the largest effect and those in Block 4 has the smallest effect. With these coefficients and setting the number of individuals $n = 300$, we generated simulation datasets for Gaussian cases.

Generating survival times and censoring indicators

In the simulation study for survival data, we need to generate the observed survival time $\widetilde{T}_i$ and the censoring indicator d_i. First, we generate survival times based on the Cox regression model, which is discussed in section 2.2.

Survival times in the Cox model can be simulated using inverse transform sampling (Bender et al., 2005). Assume that we have p-dimensional predictor vector $\boldsymbol{x}$ and coefficient vector $\boldsymbol{\beta}$. From equation (2.2.5) and equation (2.2.8), the survival function in the Cox regression is

$$S(t|\boldsymbol{x}) = \exp\left\{-A_0(t)\exp(\boldsymbol{x}\boldsymbol{\beta})\right\}, \tag{4.1.8}$$

where

$$A_0(t) = \int_0^t a_0(t)dt$$

is the cumulative baseline hazard function. Then, the cumulative distribution function $F(t|x)$ in the Cox regression is

$$F(t|\boldsymbol{x}) = 1 - S(t|\boldsymbol{x}) = 1 - \exp\left\{-A_0(t)\exp(\boldsymbol{x}\boldsymbol{\beta})\right\}. \tag{4.1.9}$$

We assume that $a_0(t) > 0$ for all $t > 0$, then $A_0(t)$ can be inverted. Let U be a random variable that follows a uniform distribution on $[0, 1]$, and let T denote the (potential) survival time in the Cox regression model. Setting $F(t|\boldsymbol{x}) = F(T) = U$,

$$T = F^{-1}(U) = A_0^{-1}\left(-\frac{\log(U)}{\exp(\boldsymbol{x}\boldsymbol{\beta})}\right). \tag{4.1.10}$$

By equation (4.1.10), we can convert uniformly distributed random numbers into survival times in the Cox model. The random numbers are available in `R`, for example, `runif(100)` can be used to generate one hundred random numbers that follow a uniform distribution on $[0, 1]$. Now, to obtain T, what remains to be done is to insert A_0^{-1}, the inverse of the cumulative baseline hazard function, into equation (4.1.10).

A common choice of the the baseline hazard function is the Weibull baseline hazard. The probability density function of the Weibull distribution is

$$f(t) = \xi\omega t^{\omega-1}e^{-\xi t^{\omega}}, \tag{4.1.11}$$

where $\xi > 0$ is called the *scale parameter* and $\omega > 0$ is the *shape parameter*. The hazard function is expressed as

$$a(t) = \xi\omega t^{\omega-1}, \tag{4.1.12}$$

which is increasing for $\omega > 1$, decreasing for $0 < \omega < 1$ and constant for $\omega = 1$. The cumulative hazard function becomes

$$A(t) = \xi t^{\omega}. \tag{4.1.13}$$

Then, the inverse of the cumulative hazard function is expressed as

$$A^{-1}(t) = (\xi^{-1}t)^{\frac{1}{\omega}}. \tag{4.1.14}$$

From equation (4.1.10) and equation (4.1.14), the survival time in the Cox model with the Weibull baseline hazard is given by

$$T = \left(\frac{-\log(U)}{\xi \exp(\boldsymbol{x}\boldsymbol{\beta})} \right)^{\frac{1}{\omega}}. \tag{4.1.15}$$

By equation 4.1.15, we can sample potential life times T for n individuals. To simulate potential censoring times C, we specify a censoring time distribution, e.g., exponential distribution or Weibull distribution. Drawing censoring times from the distribution, we obtain n pairs of potential life time and potential censoring time, (T_i, C_i) for $i = 1, \ldots, n$. Then, the observed time $\widetilde{T}_i$ is defined as $\min(T_i, C_i)$. The censoring indicator is set up such that $d_i = I(T_i \leq C_i)$, which has value 1 if the potential life time is less than or equal to the potential censoring time, i.e., 1 if the individual is not censored, and 0 otherwise. Consequently, we obtain data that has the form,

$$(\widetilde{T}_i, d_i, \boldsymbol{x}_i) \qquad \text{for} \quad i = 1, \ldots, n.$$

Setting the censoring time distribution to the exponential distribution, and setting the number of individuals $n = 300$, we generated the survival data for the simulation study. The coefficients $\boldsymbol{\beta}$ were set to the same as those in the simulation data for Gaussian cases.

4.2 Method configurations and evaluation metric

In the simulation study, we compared five methods according to the six scenarios. The methods are LBboost, priority boosting, priority-Lasso, Lasso and gradient boosting by R function `glmboost`. The specification of the five methods is given as follows.

glmboost To evaluate the performance of gradient boosting, we used function `glmboost` in R package *mboost*. Function `glmboost` implements componentwise gradient boosting that uses least-squares estimators as the base learners. The stopping number of iterations, m_{stop}, was chosen via internal 25 bootstrap iterations (default), and the step-length parameter ν was set to the default value of 0.1. More details on componentwise gradient boosting are described in section 2.1.4 and the algorithm of `glmboost` for Cox model is shown in Algorithm 2.2.1.

Lasso R package *glmnet* was used. The complexity parameter λ was chosen via internal 10-fold CV (default). Lasso is explained in Section 2.3.

priorityboost For priority boosting, R function `prioritybоost` was used, the R codes of which are shown in Appendix A.1. The priority order of the blocks was set to $(1, 2, 3, 4)$; Block 1 has the highest priority, block 2 has the second highest priority, block 3 has the third highest priority and block 4 has the lowest priority. For each block, the stopping iteration m_{stop} was chosen via internal 25 bootstrap iterations and the step-length parameter ν was set to 0.1. The option `cvoffset` is set to `FALSE` as default. For details on priority boosting, refer to Section 3.1.

prioritylasso To run priority-Lasso, R package prioritylasso was used. The priority order was set in the same way as priority boosting; Block 1 has the highest priority, block 2 has the second highest priority, block 3 has the third highest priority and block 4 has the lowest priority. For each block, the complexity parameter λ was selected by 10-fold CV. The option `cvoffset` is set to `FALSE` as default. Details on priority-Lasso is discussed in Section 2.4.

LBboost We used function `LBboost` the R codes of which are shown in Appendix A.2. The stopping iteration m_{stop} was chosen via 5-fold CV and the step-length parameter ν was set to 0.1 for Gaussian responses, and set to 0.2 for survival times to make the computation time shorter. The fitted base learners are Lasso regression models and the complexity parameter λ of the Lasso models were selected by 10-fold CV. For details on LBboost, see Section 3.2.

These methods are divided into three types in terms of how to handle the block structure in predictors;

1. Naive method: glmboost and Lasso
   Methods that do not use the block structure in predictors, i.e., They process all predictors not distinguishing types of data.
2. Priority method: priorityboost and prioritylasso
   Methods that take the block structure in predictors into account, and further define the priority order among blocks.
3. Blocked method: LBboost
   Methods that take the block structure into account, but do not assign priorities to each of the blocks.

In each scenario, as described in section 4.1.2, we generated 100 simulation datasets. Each dataset were divided into a training set and a test set. The five methods learned from the training set and built prediction models. The performances of the models that each method provided were primarily evaluated on the test set.

The prediction accuracy is assessed via the mean squared error (MSE) for Gaussian responses and the integrated Brier score (IBS) for survival times. MSE and IBS were computed in the training sets as well to evaluate possible overfitting. Overfitting occurs when random noise is incorporated into the model rather than the relationship between the predictors and the responses.

If a model is overfitting, the model explains the training data well, but the prediction accuracy for new data is low. We also compared the number of predictors selected by each method to see the sparsity of the resulting models.

4.3 Results

We show the results for Gaussian responses in Section 4.3.1 and the results for survival times in Section 4.3.2 according to the six scenarios. Table 4.1 - Table 4.12 show the summary of the prediction errors (MSE for Gaussian responses and IBS for survival times) and the number of selected predictors by the five prediction methods, priorityboost, prioritylasso, LBboost, glmboost and Lasso, besides the reference method. For Gaussian responses, as the reference method, we used the mean model, i.e., the estimation of response is the mean of the responses over all the observations in the training data, so that it use no predictors to estimate. The reference method for survival times is Kaplan-Meier Estimates, which is discussed in Section 2.2.

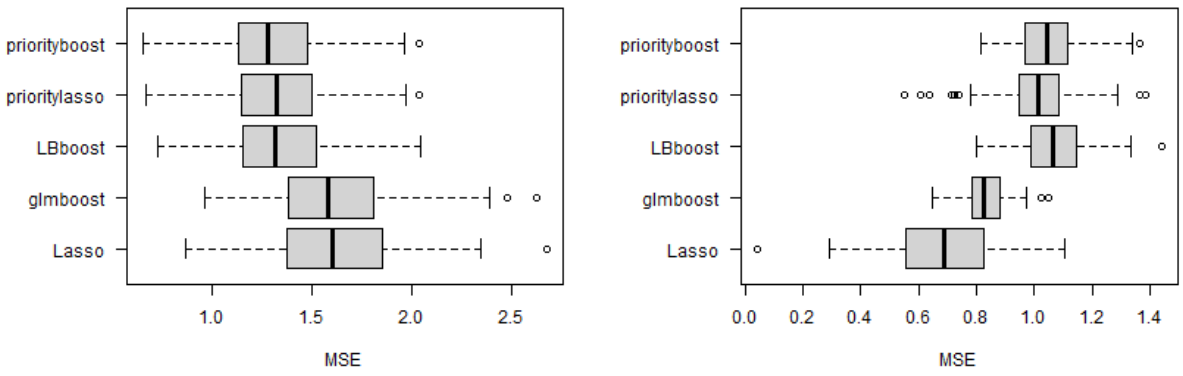
In these tables, we can see the values of the mean and the standard deviation (SD) of MSE or IBS over the test sets in columns under 'test error' and the training sets under 'training error'. Column 'total' shows the average of the total numbers of selected predictors and the subsequent columns represents the numbers of selected predictors in the respective blocks. Figure 4.1 - Figure 4.24 show the distributions of the errors or the distributions of the total number of selected predictors as box plots. In these plots, the distributions of the errors by the reference method are not included. All the five methods provided smaller errors than the reference in any scenarios, and the values by reference were sometimes too much larger to keep the visibility of the plots.

4.3.1 Part I. Simulation for Gaussian responses

Scenario 1. LLLH, Independent variables

Table 4.1. Summary of MSE and selected predictors in Scenario 1 for Gaussian responses

	test error		training error		selected predictors				
Method	Mean	SD	Mean	SD	ALL	Block 1	Block 2	Block 3	Block 4
priorityboost	1.312	0.260	1.046	0.111	23.99	7.82	7.64	7.64	0.89
prioritylasso	1.339	0.262	1.007	0.142	28.95	8.03	7.92	7.93	5.07
LBboost	1.347	0.262	1.067	0.111	50.44	9.08	9.06	8.78	23.52
glmboost	1.636	0.340	0.831	0.075	58.33	5.21	5.22	4.99	42.91
Lasso	1.631	0.343	0.683	0.222	83.95	5.33	5.34	5.19	68.09
Reference	7.817	1.289	8.008	0.788	-	-	-	-	-



(a) test error (b) training error

Figure 4.1: MSE in Scenario 1 for Gaussian responses

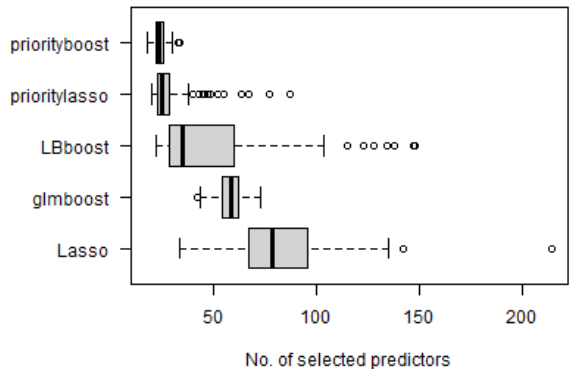
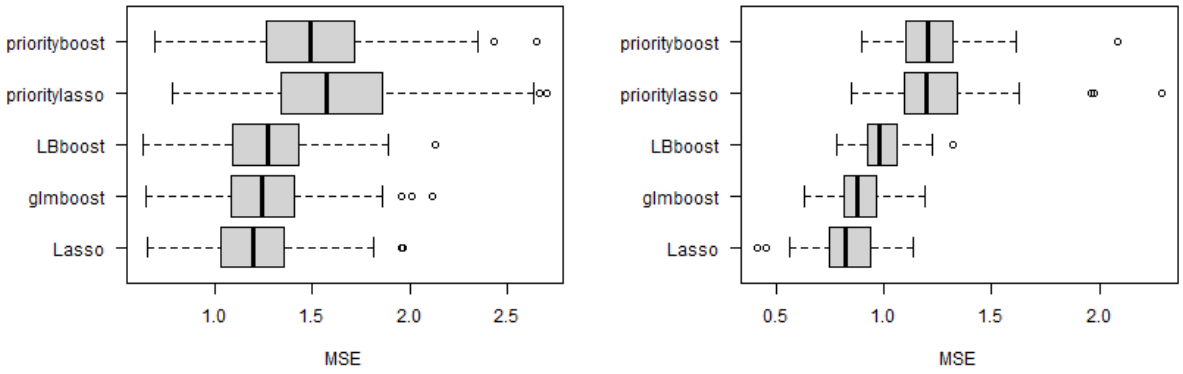


Figure 4.2: Total number of selected predictors in Scenario 1 for Gaussian responses

Part I. Scenario 2. LLLH, Dependencies within each group

Table 4.2. Summary of MSE and selected predictors in Scenario 2 for Gaussian responses

	test error		training error		selected predictors				
Method	Mean	SD	Mean	SD	ALL	Block 1	Block 2	Block 3	Block 4
prioritylasso	1.511	0.367	1.219	0.179	24.38	6.22	6.45	5.83	5.88
prioritylasso	1.609	0.405	1.229	0.237	28.36	6.54	6.46	5.96	9.40
LBboost	1.275	0.259	0.994	0.111	45.97	8.36	8.21	7.46	21.94
glmboost	1.266	0.268	0.892	0.114	35.56	5.57	5.63	5.13	19.23
Lasso	1.209	0.249	0.832	0.147	40.90	5.66	5.67	5.27	24.30
Reference	30.014	5.130	30.805	1.976	-	-	-	-	-



(a) test error (b) training error

Figure 4.3: MSE in Scenario 2 for Gaussian responses

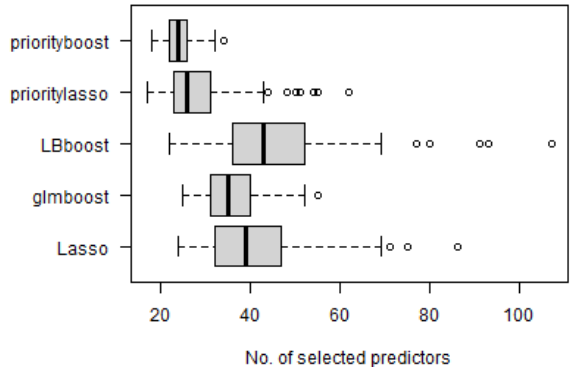
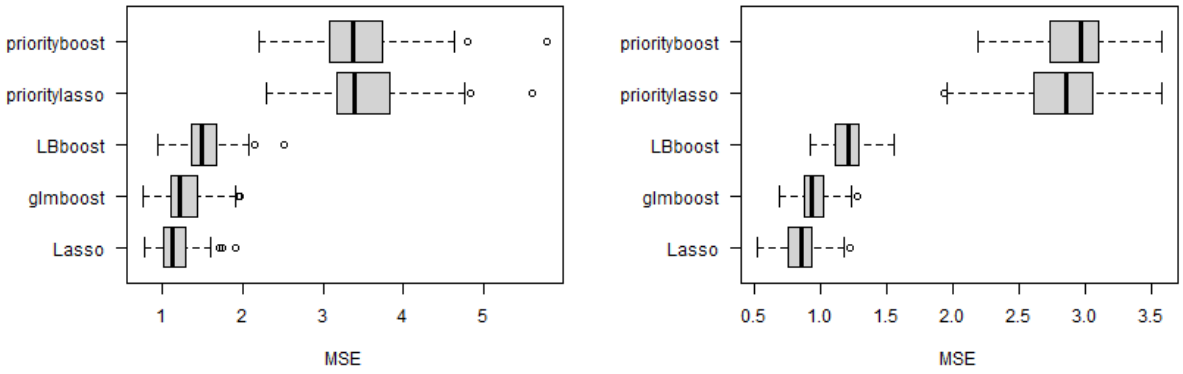


Figure 4.4: Total number of selected predictors in Scenario 2 for Gaussian responses

Part I. Scenario 3. LLLH, Dependencies among groups

Table 4.3. Summary of MSE and selected predictors in Scenario 3 for Gaussian responses

	test error		training error		selected predictors				
Method	Mean	SD	Mean	SD	ALL	Block 1	Block 2	Block 3	Block 4
prioritylasso	3.455	0.600	2.928	0.294	22.72	7.63	7.37	6.19	1.53
prioritylasso	3.488	0.599	2.838	0.341	25.51	7.40	7.64	6.64	3.83
LBboost	1.521	0.261	1.204	0.127	44.63	9.02	9.01	7.74	18.86
glmboost	1.277	0.257	0.952	0.116	28.30	5.95	5.89	5.33	11.13
Lasso	1.165	0.221	0.848	0.133	35.68	6.49	6.12	5.71	17.36
Reference	47.073	7.812	48.154	3.004	-	-	-	-	-



(a) test error (b) training error

Figure 4.5: MSE in Scenario 3 for Gaussian responses

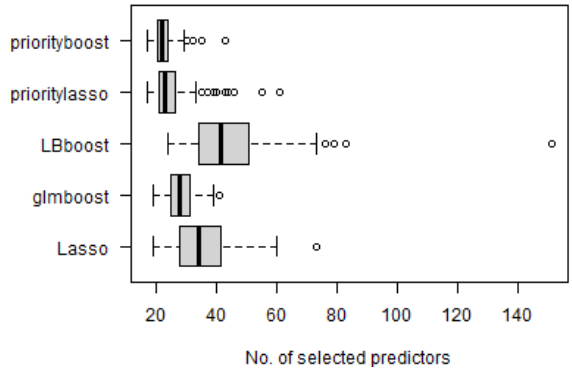
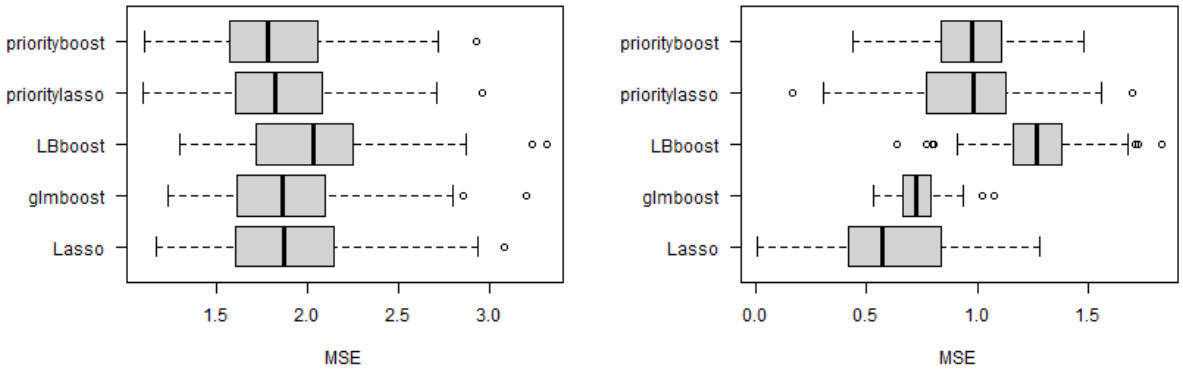


Figure 4.6: Total number of selected predictors in Scenario 3 for Gaussian responses

Part I. Scenario 4. LHHH, Independent variables

Table 4.4. Summary of MSE and selected predictors in Scenario 4 for Gaussian responses

	test error		training error		selected predictors				
Method	Mean	SD	Mean	SD	ALL	Block 1	Block 2	Block 3	Block 4
priorityboost	1.825	0.351	0.971	0.216	56.07	7.89	34.01	13.56	0.61
prioritylasso	1.859	0.357	0.954	0.288	69.10	8.04	32.87	24.45	3.74
LBboost	2.038	0.398	1.272	0.212	136.00	8.89	56.47	49.41	21.23
glmboost	1.889	0.384	0.736	0.098	75.19	5.15	24.74	24.82	20.48
Lasso	1.897	0.382	0.610	0.282	105.57	5.20	34.95	34.74	30.68
Reference	7.779	1.408	8.088	0.670	-	-	-	-	-



(a) test error (b) training error

Figure 4.7: MSE in Scenario 4 for Gaussian responses

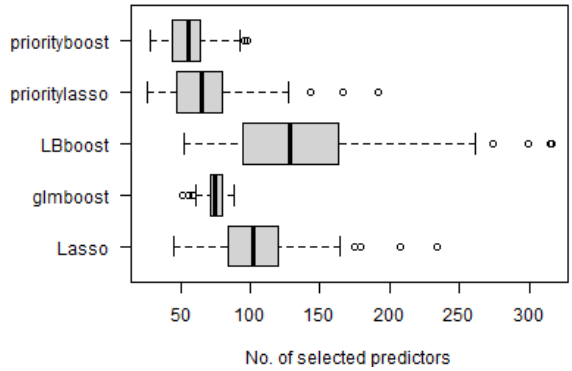
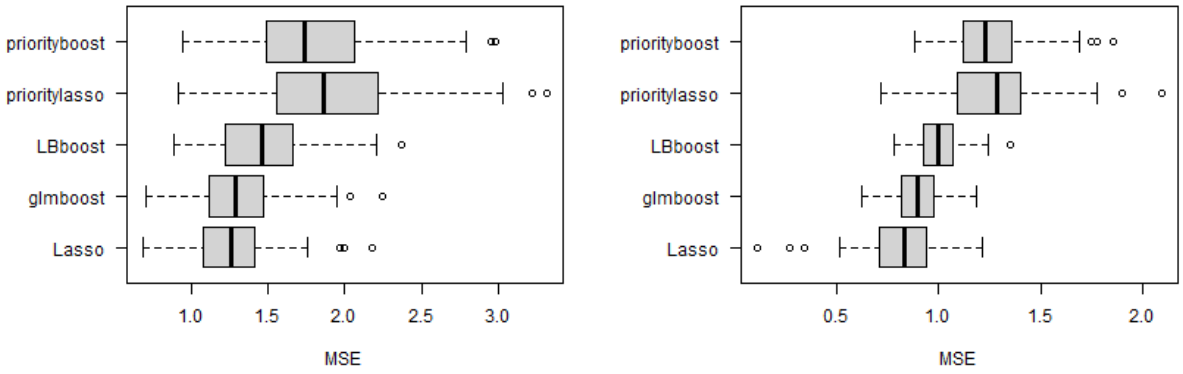


Figure 4.8: Total number of selected predictors in Scenario 4 for Gaussian responses

Part I. Scenario 5. LHHH, Dependencies within each group

Table 4.5. Summary of MSE and selected predictors in Scenario 5 for Gaussian responses

	test error		training error		selected predictors				
Method	Mean	SD	Mean	SD	ALL	Block 1	Block 2	Block 3	Block 4
priorityboost	1.784	0.419	1.246	0.198	31.74	5.97	10.26	9.51	6.00
prioritylasso	1.900	0.473	1.271	0.243	37.00	6.00	11.57	10.54	8.89
LBboost	1.454	0.297	0.999	0.113	87.66	8.04	32.00	27.68	19.94
glmboost	1.310	0.290	0.901	0.114	36.36	5.48	11.22	10.67	8.99
Lasso	1.258	0.273	0.812	0.181	48.43	5.54	15.35	14.71	12.83
Reference	29.733	5.096	30.621	2.224	-	-	-	-	-



(a) test error (b) training error

Figure 4.9: MSE in Scenario 5 for Gaussian responses

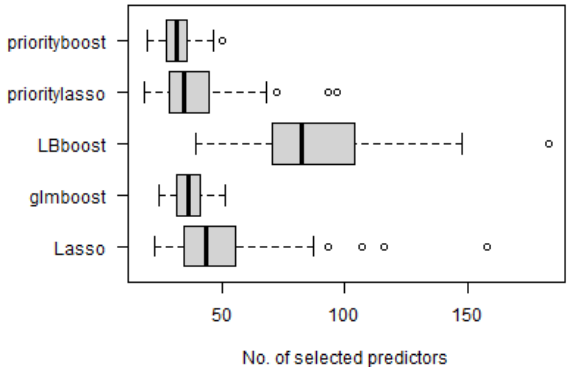
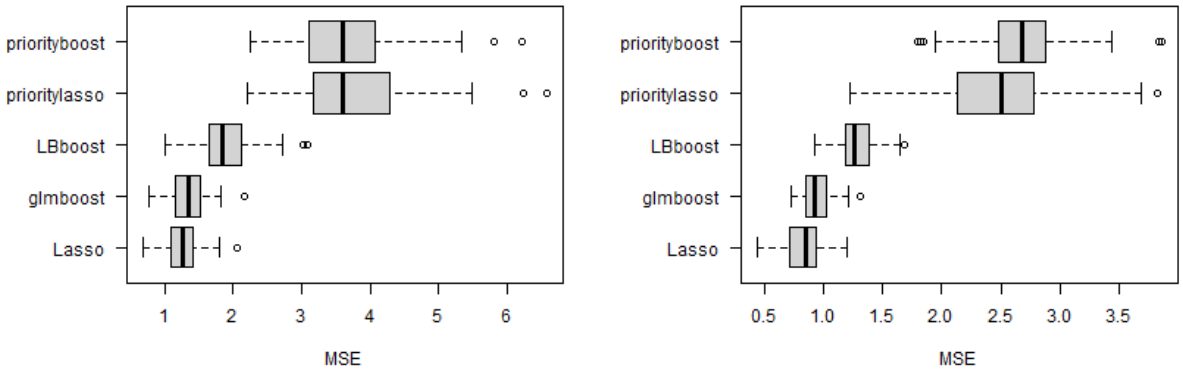


Figure 4.10: Total number of selected predictors in Scenario 5 for Gaussian responses

Part I. Scenario 6. LHHH, Dependencies among groups

Table 4.6. Summary of MSE and selected predictors in Scenario 6 for Gaussian responses

	test error		training error		selected predictors				
Method	Mean	SD	Mean	SD	ALL	Block 1	Block 2	Block 3	Block 4
prioritylasso	3.677	0.736	2.675	0.373	30.36	7.90	11.97	8.15	2.34
prioritylasso	3.756	0.839	2.453	0.496	42.44	7.65	18.64	10.67	5.48
LBboost	1.873	0.385	1.287	0.167	92.93	8.91	36.31	28.34	19.37
glmboost	1.359	0.253	0.948	0.126	32.24	5.94	9.95	9.19	7.16
Lasso	1.262	0.238	0.836	0.164	43.82	6.13	13.49	13.12	11.08
Reference	41.280	7.180	42.094	2.932	-	-	-	-	-



(a) test error (b) training error

Figure 4.11: MSE in Scenario 6 for Gaussian responses

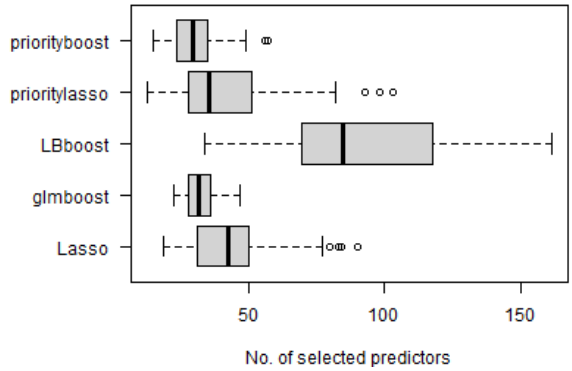


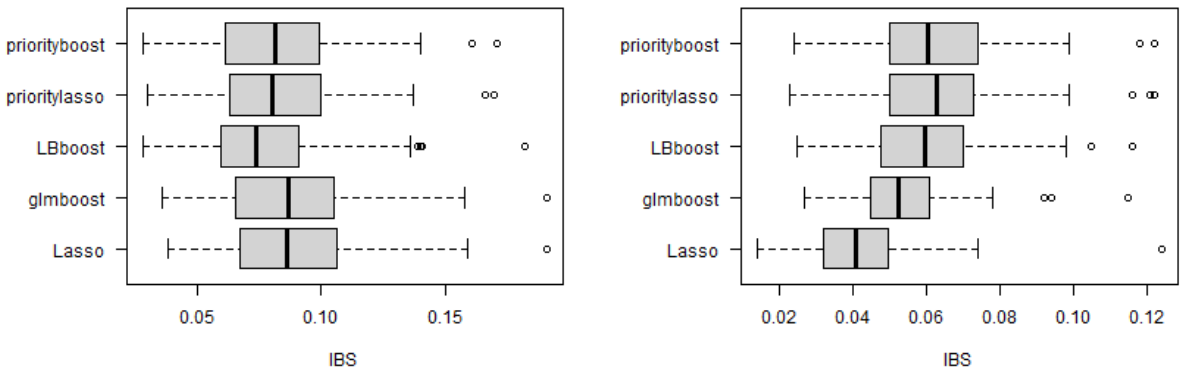
Figure 4.12: Total number of selected predictors in Scenario 6 for Gaussian responses

4.3.2 Part II. Simulation for survival times

Scenario 1. LLLH, Independent variables

Table 4.7. Summary of IBS and selected predictors in Scenario 1 for survival times

	test error		training error		selected predictors				
Method	Mean	SD	Mean	SD	ALL	Block 1	Block 2	Block 3	Block 4
priorityboost	0.0831	0.0285	0.0633	0.0177	22.68	7.51	7.41	6.86	0.90
prioritylasso	0.0834	0.0287	0.0626	0.0179	24.58	7.63	7.50	7.46	1.99
LBboost	0.0775	0.0269	0.0596	0.0167	33.74	8.94	8.67	8.39	7.74
glmboost	0.0882	0.0289	0.0536	0.0143	33.18	5.18	5.10	2.71	20.19
Lasso	0.0885	0.0284	0.0429	0.0152	55.68	5.28	5.17	3.60	41.63
Reference	0.1877	0.0208	0.1738	0.0178	-	-	-	-	-



(a) test error (b) training error

Figure 4.13: IBS in Scenario 1 for survival times

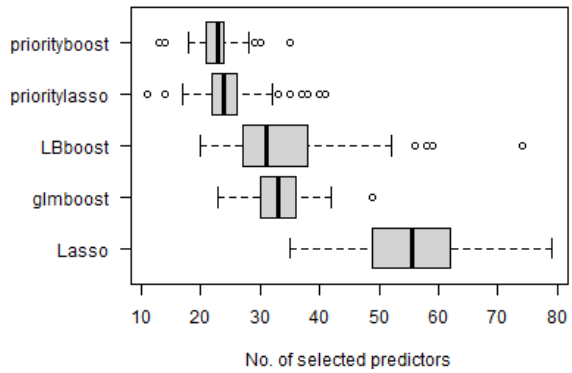
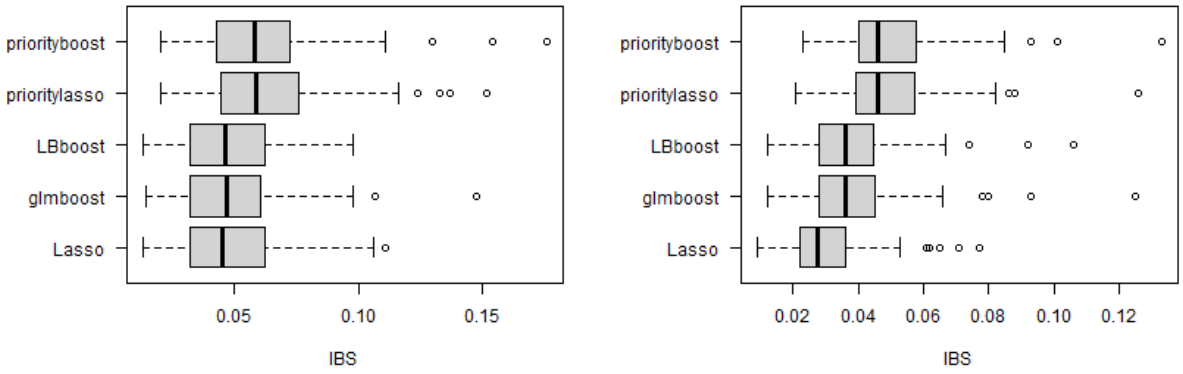


Figure 4.14: Total number of selected predictors in Scenario 1 for survival times

Part II. Scenario 2. LLLH, Dependencies within each group

Table 4.8. Summary of IBS and selected predictors in Scenario 2 for survival times

	test error		training error		selected predictors				
Method	Mean	SD	Mean	SD	ALL	Block 1	Block 2	Block 3	Block 4
priorityboost	0.0604	0.0271	0.0502	0.0176	17.39	5.27	5.74	4.43	1.95
prioritylasso	0.0614	0.0264	0.0490	0.0166	19.63	5.34	5.75	4.77	3.77
LBboost	0.0482	0.0204	0.0381	0.0151	33.88	7.91	7.58	6.57	11.82
glmboost	0.0486	0.0223	0.0388	0.0163	19.46	5.49	5.34	4.00	4.63
Lasso	0.0485	0.0209	0.0303	0.0126	45.13	5.67	5.66	4.50	29.30
Reference	0.2273	0.0108	0.2206	0.0119	-	-	-	-	-



(a) test error (b) training error

Figure 4.15: IBS in Scenario 2 for survival times

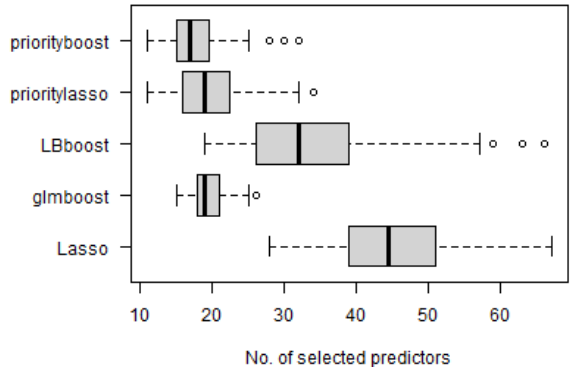
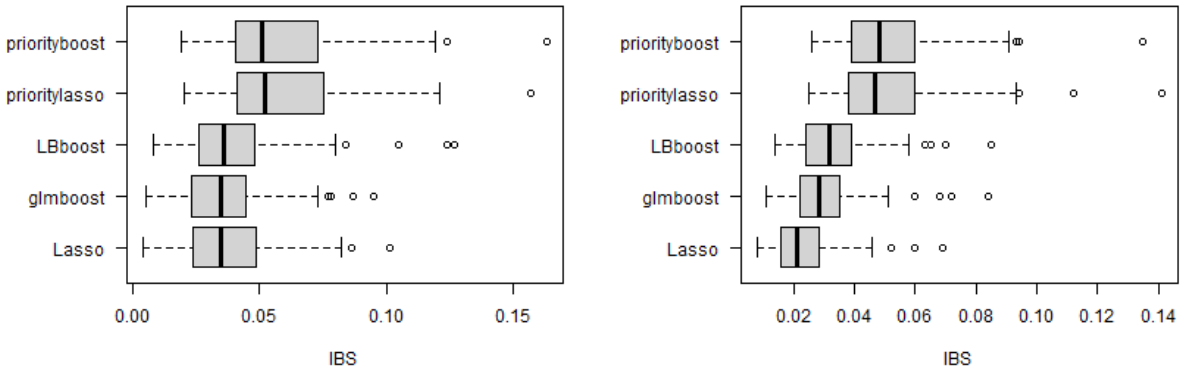


Figure 4.16: Total number of selected predictors in Scenario 2 for survival times

Part II. Scenario 3. LLLH, Dependencies among groups

Table 4.9. Summary of IBS and selected predictors in Scenario 3 for survival times

	test error		training error		selected predictors				
Method	Mean	SD	Mean	SD	ALL	Block 1	Block 2	Block 3	Block 4
priorityboost	0.0576	0.0251	0.0523	0.0181	17.73	6.77	5.96	4.21	0.79
prioritylasso	0.0574	0.0245	0.0515	0.0194	19.44	6.73	6.00	4.71	2.00
LBboost	0.0404	0.0216	0.0340	0.0125	33.88	7.91	7.58	6.57	11.82
glmboost	0.0367	0.0179	0.0301	0.0121	21.33	6.49	5.91	4.78	4.15
Lasso	0.0377	0.0184	0.0231	0.0107	46.00	6.41	5.92	4.90	28.77
Reference	0.2339	0.0079	0.2293	0.0095	-	-	-	-	-



(a) test error (b) training error

Figure 4.17: IBS in Scenario 3 for survival times

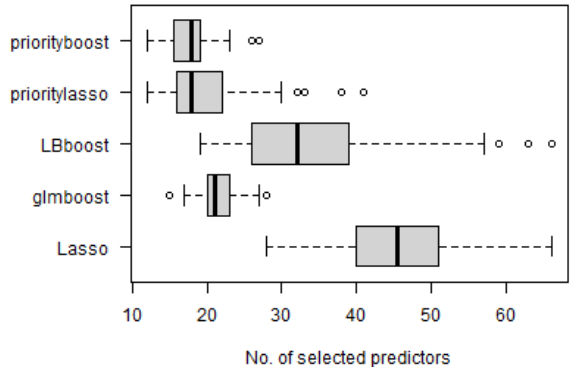
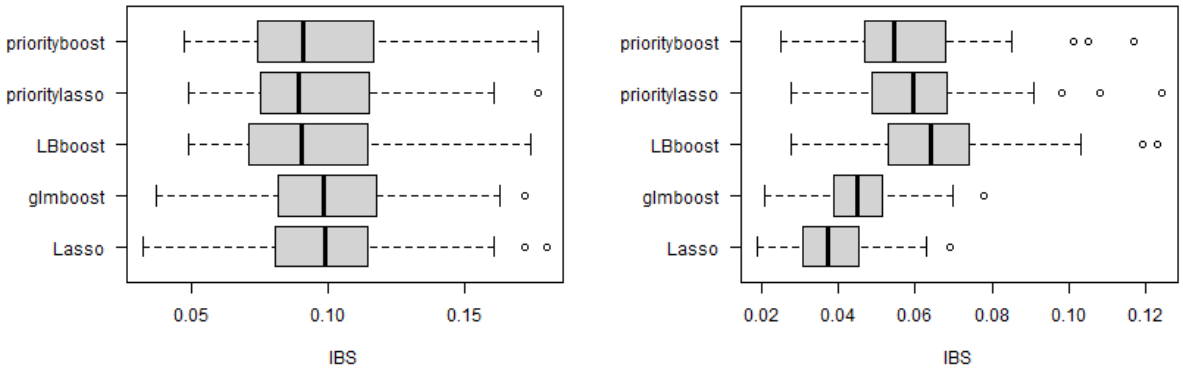


Figure 4.18: Total number of selected predictors in Scenario 3 for survival times

Part II. Scenario 4. LHHH, Independent variables

Table 4.10. Summary of IBS and selected predictors in Scenario 4 for survival times

	test error		training error		selected predictors				
Method	Mean	SD	Mean	SD	ALL	Block 1	Block 2	Block 3	Block 4
priorityboost	0.0970	0.0288	0.0578	0.0163	35.70	7.58	23.90	3.26	0.96
prioritylasso	0.0968	0.0288	0.0604	0.0163	35.38	7.64	21.03	4.81	1.90
LBboost	0.0960	0.0298	0.0642	0.0164	62.92	8.57	33.56	13.72	7.07
glmboost	0.1003	0.0278	0.0460	0.0107	42.98	5.05	15.10	12.25	10.58
Lasso	0.1002	0.0283	0.0392	0.0108	60.89	5.09	20.59	18.47	16.74
Reference	0.1891	0.0187	0.1706	0.0189	-	-	-	-	-



(a) test error (b) training error

Figure 4.19: IBS in Scenario 4 for survival times

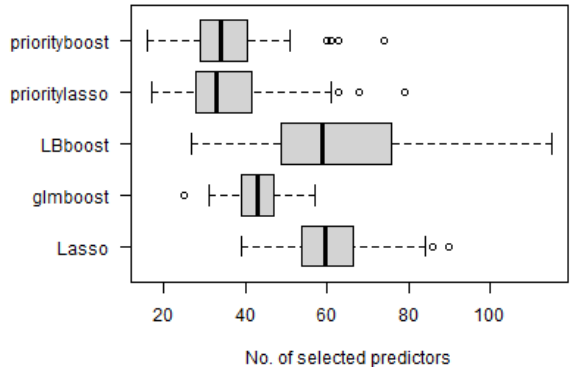
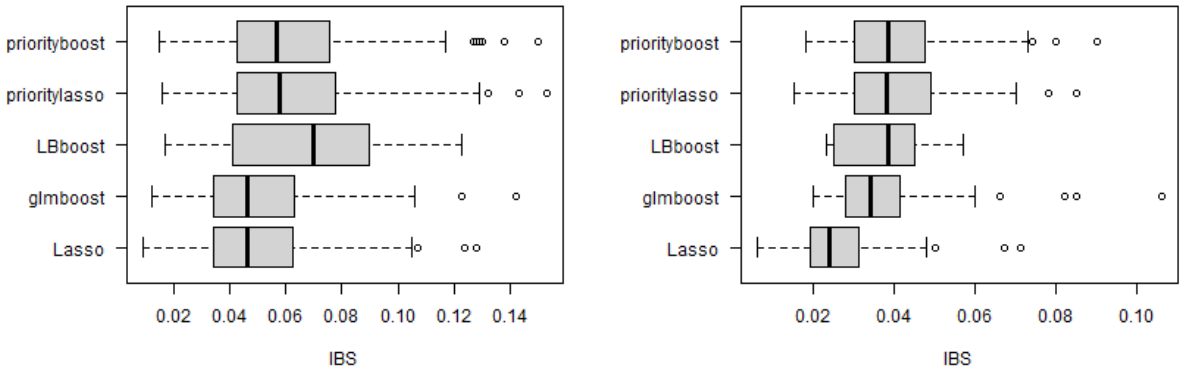


Figure 4.20: Total number of selected predictors in Scenario 4 for survival times

Part II. Scenario 5. LHHH, Dependencies within each group

Table 4.11. Summary of IBS and selected predictors in Scenario 5 for survival times

	test error		training error		selected predictors				
Method	Mean	SD	Mean	SD	ALL	Block 1	Block 2	Block 3	Block 4
priorityboost	0.0629	0.0294	0.0412	0.0147	27.46	5.30	12.24	7.94	1.98
prioritylasso	0.0634	0.0304	0.0406	0.0141	27.17	5.37	10.81	7.43	3.56
LBboost	0.0656	0.0331	0.0371	0.0115	62.98	7.59	24.65	19.25	11.49
glmboost	0.0506	0.0249	0.0368	0.0141	24.13	5.44	7.92	6.52	4.25
Lasso	0.0515	0.0260	0.0261	0.0109	54.54	5.52	18.29	16.30	14.43
Reference	0.2250	0.0127	0.2191	0.0113	-	-	-	-	-



(a) test error (b) training error

Figure 4.21: IBS in Scenario 5 for survival times

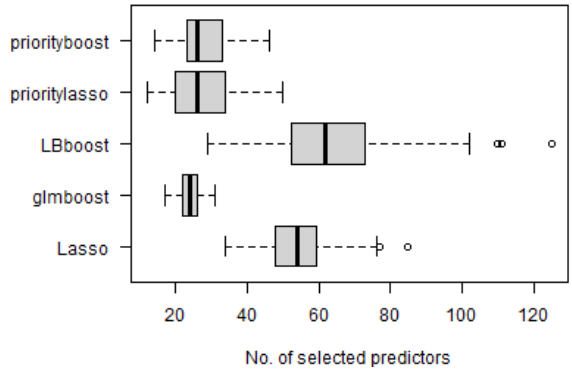
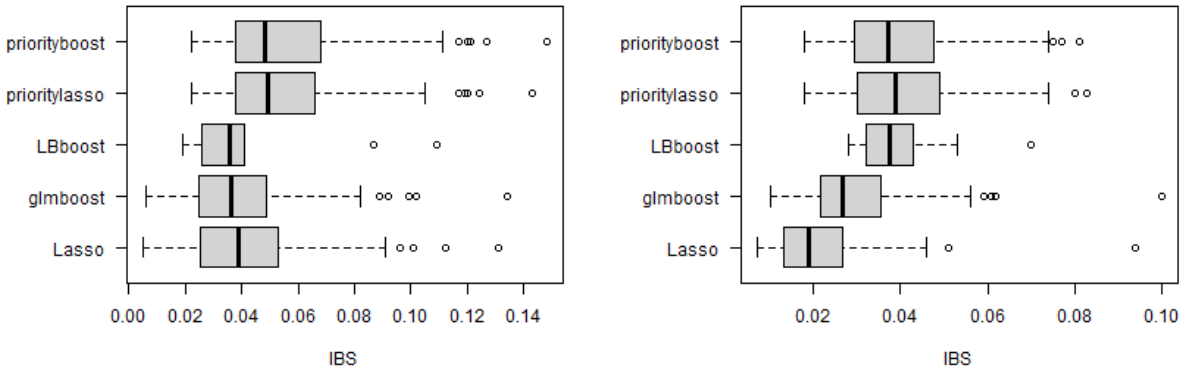


Figure 4.22: Total number of selected predictors in Scenario 5 for survival times

Part II. Scenario 6. LHHH, Dependencies among groups

Table 4.12. Summary of IBS and selected predictors in Scenario 6 for survival times

	test error		training error		selected predictors				
Method	Mean	SD	Mean	SD	ALL	Block 1	Block 2	Block 3	Block 4
priorityboost	0.0556	0.0261	0.0400	0.0141	36.37	7.07	20.10	7.34	1.86
prioritylasso	0.0555	0.0260	0.0407	0.0147	35.68	6.78	16.71	8.46	3.73
LBboost	0.0443	0.0283	0.0408	0.0120	67.35	8.16	30.65	16.19	12.35
glmboost	0.0417	0.0223	0.0307	0.0136	24.67	6.48	7.95	6.34	3.90
Lasso	0.0431	0.0230	0.0215	0.0118	55.19	6.28	18.69	15.94	14.28
Reference	0.2318	0.0105	0.2269	0.0106	-	-	-	-	-



(a) test error

(b) training error

Figure 4.23: IBS in Scenario 6 for survival times

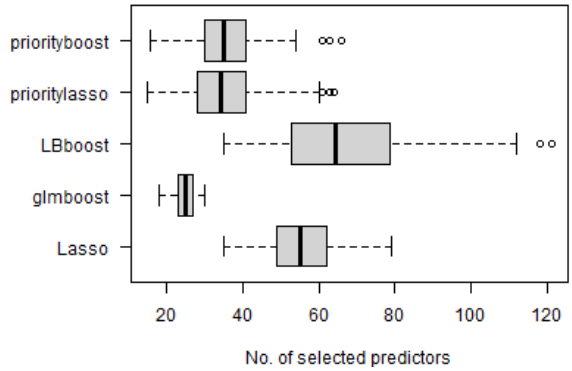


Figure 4.24: Total number of selected predictors in Scenario 6 for survival times

4.4 Discussion

We discuss our findings from the results in six key points. Firstly, in most of the scenarios for both Gaussian responses and survival times, the two priority methods, priorityboost and prioritylasso, selected fewer predictors than other three methods. The priority methods provided sparser models. In contrast, Lasso and LBboost often selected more predictors than other three methods. The number of predictors selected by glmboost varied across scenarios, but it was always fewer than predictors by Lasso and sometimes even fewer than those by the prioritylasso.

Secondly, the two priority methods tended to select more predictors from blocks with higher priorities than from blocks with lower priorities. Let us consider the two settings with dependencies. In the scenarios where there are dependencies among blocks (Scenario 3 and 6), the total number of predictors chosen from Block 1-3 increased, and the number of predictors from Block 4 decreased in most cases, compared with the scenarios where there are dependencies within each block (Scenario 2 and 5). This may be from the hierarchical structure in the priority methods; Predictors from blocks with low priorities are incorporated in the model only if they explain variability that cannot be explained by blocks with higher priority. This result is consistent with description of priority-Lasso in Klau et al. (2018), "An important feature of priority-Lasso is that it directly addresses the problem of redundancies in the predictive information across different blocks".

Thirdly, the naive methods, especially Lasso, showed a tendency toward overfitting in the settings where all variables are independent. In the results of Scenario 1 and 4, Lasso provided the smallest training error, but the largest or the second largest test error. We can see that Lasso selected more predictors in Block 4, which was assigned the lowest priority, than other methods. For example, in Scenario 1 for Gaussian responses, the average number of predictors selected from Block 4 by prioritylasso is 5.07 and that by Lasso is 68.09. Note that in our simulation design, we assume that there are five predictors that have effects on the responses in Block 4. These results suggest that Lasso was unable to distinguish the effects of predictors from noise and selected more irrelevant predictors than other methods, leading to overfitting. The results from glmboost, while not as pronounced as those from Lasso, show a similar trend.

On the other hand, the results of other three methods did not show the tendency of overfitting. It is not surprising for the priority methods not to be overfitting, considering that they selected fewer predictors. Let us discuss LBboost with the above example, Scenario 1 for Gaussian responses. The average number of selected predictors from Block 4 by LBboost is 23.52, which is fewer than Lasso. As discussed in Section 3.2, in each boosting iteration, LBboost compares the candidate updates by each block and incorporates only the effects of predictors from the block that provided the best update. It seems that this screening enabled the prediction models to avoid having too much irrelevant predictors from Block 4 unlike the naive methods.

Fourthly, in the scenarios where there are dependencies regardless within

each block or among blocks, i.e, in Scenario 2, 3, 5 and 6, LBboost and the two naive methods tended to provide lower test errors than the priority methods. By contrast with the results in Scenario 1 or 4, Lasso and glmboost seems to work well there in terms of prediction accuracy.

As we see in Section 4.1.2, the simulation data were generated assuming that five predictors in each block have effect on the responses. However, in these scenarios, due to the correlation structures in the predictors, there are more than five predictors that are relevant to the responses. The "effects" of the correlated predictors are "spurious", not effect itself, but they still have ability to describe some part of the variability in the responses. One possibility of the reason why the naive methods worked better in these scenarios is that the number of relevant predictors selected by these methods increased than Scenario 1 or 4, where it is seemed that they selected more predictors by random noises.

In terms of prediction accuracy, the priority methods have advantages and disadvantages. In this simulation study, we generated the data assuming that predictors in Block 1 has the largest effect and those in Block 4 has the smallest effect. The priority methods incorporate this "prior knowledge" via setting the priority order among blocks. On the other hand, due to the hierarchical structure, these methods might underestimate the influences of blocks with lower priority. There is possibility that they eliminate predictors that have effect in blocks with lower priority, which leads to sparser model, but also makes their prediction accuracy worse. This might be one reason why the test errors by the priority methods were larger than the other three methods in these scenarios. (Note that function `prioritylasso` and `priorityboost` have option `cvoffset` to avoid the influences on the prediction accuracy due to underestimating the impact by blocks with lower priorities, which is discussed in 2.4.3. In this simulation study, we did not use this option as mentioned in Section 4.2.)

Fifthly, in most of the scenarios, LBboost provided relatively smaller test errors. LBboost and the priority methods often gave smaller test errors than the naive methods in scenarios where all variables are independent (Scenario 1 and 4), whereas LBboost and the naive methods tended to provide smaller test errors than the priority methods in scenarios where there are dependencies (Scenario 2, 3, 5 and 6). The exceptions are Scenario 4 for Gaussian responses and Scenario 5 for survival times, where the mean value of the test errors by LBboost was the largest of the five methods. However, it may be assumed that the differences among the methods are not large in these scenarios. We can say that LBboost had no scenarios that yield considerably larger test errors compared to the other four methods.

Finally, we compare the results of priorityboost with those of prioritylasso. In most of the scenarios, priorityboost selected fewer predictors and provided smaller test errors than prioritylasso. However, the differences seem to be small and it can be seen that their results are the almost same regarding both prediction accuracy and sparsity of the resulting model.

CHAPTER 5

Application to real multi-omics data

In this chapter, we evaluate the performances of priority boosting and Lasso-based block boosting (LBboost) on a real multi-omics dataset. The dataset is regarding acute myeloid leukemia (AML) patients, which is described in Section 5.1. AML is a heterogeneous disease characterized by a large number of cytogenetic and molecular genetic aberrations, which result in significant differences in responses and survival after treatment among the patients. On the dataset, we compare the results by the above two methods with the other three methods, priority-Lasso, Lasso and gradient boosting. We assess the prediction accuracy via the integrated Brier score, and the sparsity of the resulting models.

5.1 Acute myeloid leukemia data

We consider the dataset consisting of AML patients collected in Germany. Hereinafter, we call it the AML dataset. The patients were treated in the multicenter randomized phase III trial (clinicaltrials.gov identifier NCT00266136) by the German Acute Myeloid Leukemia Cooperative Group (AMLCG) between 1999 and 2005 (Büchner et al., 2006). The gene expression data corresponding to the patients can be found in genomics data repository Gene Expression Omnibus (GSE37642). The outcome of the dataset is the overall survival time and patients with translocation t(15;17) or myelodysplastic syndrome are excluded (Klau et al., 2018). Also, patients with missing values are excluded, resulting in 359 observations. The number of the observation with events is 257 and the number of the censored observations is 102. The dataset consists of 44812 predictors; clinical variables, cytogenetics, gene mutations and gene expression variables.

In the following sections, we compared five prediction methods on the dataset. Priority boosting, priority-Lasso and Lasso-based block boosting are methods that take the block structure of the variables into account. The block structure on the AML dataset were defined as follows.

- Block 1: the three-categorical MRC score. It is represented by 2 dummy variables.
- Block 2: 8 clinical variables measured on different scales, binary or continuous.

- Block 3: 48 binary variables, each representing the mutation status for a certain gene.
- Block 4: 44754 continuous variables, each representing the expression value of a certain gene.

Block 1 consists of two dummy variables which represent three risk groups of AML patients, favorable, intermediate and adverse. This risk groups were defined on the basis of certain cytogenetic aberrations according to 2010 UK Medical Research Council (MRC) classification (Grimwade et al., 2010).

Priority boosting and priority-Lasso require to define the priority order among blocks on data. We set the priority order to $(1, 2, 3, 4)$; Block 1 has the highest priority, block 2 has the second highest priority, block 3 has the third highest priority and block 4 has the lowest priority.

5.2 Method configurations and evaluation metric

On the AML data, we evaluated the performance of the five methods, which are the same in the simulation study in Chapter 4; LBboost, priority boosting, priority-Lasso, gradient boosting and Lasso. The specification of these five methods is listed below.

glmboost In common with the simulation study, to assess the performance of gradient boosting, we used function `glmboost` in R package mboost, which implements the componentwise gradient boosting that uses least-squares estimators as the base learners. The stopping iteration m_{stop} was chosen via internal 25 bootstrap iterations (default), and the step-length parameter ν was set to the default value of 0.1. The componentwise gradient boosting approach is discussed in section 2.1.4 and the algorithm of `glmboost` for the Cox model is described in algorithm 2.2.1.

Lasso R package *glmnet* was used. The complexity parameter λ was chosen via internal 10-fold CV (default). Lasso is explained in Section 2.3.

prioritybоost Priority boosting is implemented by function `prioritybоost`. The R codes are shown in Appendix A and details on priority boosting are discussed in Section 3.1. For each block, the stopping iteration m_{stop} was selected via internal 25 bootstrap iterations, and the step-length parameter ν was set to 0.1. The option `cvoffset` is set to `FALSE` as default.

prioritylasso For priority-Lasso, R package *prioritylasso* was used. For each block, the complexity parameter λ was chosen by 10-fold CV. The option `cvoffset` is set to `FALSE` as default. For details on priority-Lasso, see in Section 2.4.

LBboost We used function `LBboost`. The R codes are shown in Appendix A. The stopping iteration m_{stop} was chosen via 5-fold CV and the step-length parameter ν was set to 0.2 to make the computation time shorter. The base learners of LBboost are Lasso regression models and the complexity parameter

λ of the Lasso models were selected by 10-fold CV. For details on LBboost, refer to Section 3.2.

In common with the simulation study, these five methods are divided into three types in terms of how to handle the block structure in predictors; 1) Naive method: glmboost and Lasso, 2) Priority method: prioritylasso and prioritylasso, 3) Blocked method: LBboost. Description of the three type is in Section 4.2.

To compare the prediction performance of these methods, we performed five repetitions of 5-fold cross-validation. Hence, each method runs $5 \cdot 5 = 25$ times. The prediction performances were assessed via the integrated Brier score (IBS). IBS were computed in the training sets as well as the test set. To see the sparsity of the resulting models, we compared the number of the predictors chosen by each method.

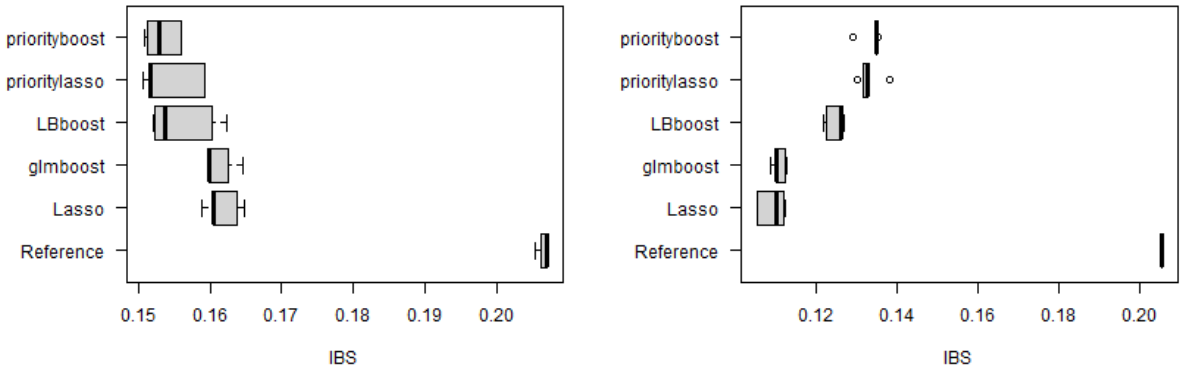
5.3 Results

Table 5.1 shows the summary of IBS and the number of selected predictors by the five methods, prioritylasso, prioritylasso, LBboost, glmboost and Lasso, besides the reference method. These values are evaluated on five repetitions of 5-fold crossvalidation (CV). The reference method is Kaplan-Meier estimate, which is discussed in Section 2.2.

In this table, we can see the values of the mean and the standard deviation (SD) of IBS over the test sets in columns under 'test error' and the training sets under 'training error'. Column 'total' shows the average of the total numbers of selected predictors and the subsequent columns represents the numbers of selected predictors in the respective blocks. Figure 5.1 displays the distributions of the test errors and the training errors as box plots. Figure 5.2 presents the distribution of the total number of selected predictors.

Table 5.1. Summary of IBS and selected predictors on AML data

	test error		training error		selected predictors				
Method	Mean	SD	Mean	SD	ALL	Block 1	Block 2	Block 3	Block 4
prioritylasso	0.1534	0.0025	0.1335	0.0026	19.24	2.00	5.96	2.44	8.84
prioritylasso	0.1544	0.0044	0.1330	0.0031	24.68	2.00	5.80	3.48	13.40
LBboost	0.1561	0.0049	0.1246	0.0024	50.92	1.28	6.80	5.72	37.12
glmboost	0.1613	0.0022	0.1106	0.0017	33.72	0.32	1.84	0.32	31.24
Lasso	0.1616	0.0025	0.1088	0.0034	38.88	0.24	1.96	0.40	36.28
Reference	0.2065	0.0007	0.2054	0.0000	-	-	-	-	-



(a) test errors (b) training errors

Figure 5.1: IBS on AML data

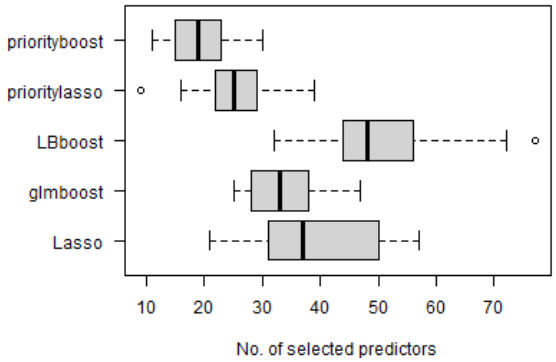


Figure 5.2: Total number of selected predictors on AML data

5.4 Discussion

In the results, we can see that all the five methods provided smaller mean values of test errors than the Kaplan-Meier estimate. Let us check the number of selected predictors. The two priority methods selected fewer predictors than other methods, which is consistent with the result of the simulation study in Chapter 4. Block 1, which was assigned the highest priority, consists of two variables, and the priority methods selected both of these variables over all the 25 runs. From Block 4, which consists of 44754 variables, prioritylasso chose 8.84 predictors in average and prioritylasso 13.40 predictors, which are fewer than other three methods. This could be the result of setting the priority order among the blocks.

The mean values of test errors by the priority methods and LBboost are smaller than those by the naive methods, Lasso and glmboost. On the AML data, the priority methods seem to have worked well in terms of both prediction accuracy and model sparsity, taking advantage of prior knowledge. It should be noted that the naive methods, Lasso and glmboost, appear to overfit the training data. Two variables in Block 1 are known to yield good prediction

accuracy. However, the average number of selected predictors from this block by these naive methods were fewer than the other methods, i.e, 0.32 by glmboost and 0.24 by Lasso. On the other hand, these methods chose more than 30 predictors from Block 4 in average. Considering that these methods provided smaller training errors, but larger test errors than the other three methods, it seems that they selected irrelevant predictors that do not explain the variability of the responses.

In contrast, the results of LBboost show no obvious sign of overfitting, which is consistent with the result of the simulation study. The number of selected predictors by LBboost is larger than those by the other methods. It seems that while the naive methods tended to select irrelevant predictors, LBboost selected more relevant predictors than irrelevant ones. This result is consistent with the findings of the simulation study in Chapter 4 and this may be caused by the subset-updating approach in LBboost. We can see that the mean value of the test errors by LBboost is as small as those by the priority methods. Note that LBboost did not use the prior knowledge and derive the estimates purely data-driven.

CHAPTER 6

Conclusion and future work

6.1 Conclusion

In this thesis, we introduced priority boosting and Lasso-based block boosting (LBboost), which are two novel approaches to build regression models for datasets that contain different types of data, such as multi-omics data. We evaluated their prediction abilities on simulation data and a real multi-omics dataset, the AML data, comparing them with three other methods, priority-Lasso, Lasso and glmboost.

The results in Section 4 and 5 show the following characteristics of priority boosting. The resulting prediction models tend to be sparser. They often have a smaller number of predictors compared to Lasso, glmboost and LBboost. In addition, priority boosting favors predictors of blocks with higher priorities, i.e., particular "favorite" blocks, over predictors in blocks with lower priorities. This property might affect on the prediction accuracy positively and negatively, as discussed in Section 4.4. Namely, incorporating prior knowledge by setting favorite groups helps the model to enhance prediction ability, while underestimating the influences of blocks with lower priorities might lead to larger error.

In the simulation study, the priority methods showed larger test errors than the other methods in some scenarios. However, in the application to the AML data, priority boosting was able to provide the best prediction accuracy of the five methods, taking advantage of prior knowledge. Through all the simulation datasets and the AML dataset, priority-Lasso presented similar results to priority boosting, in terms of prediction accuracy and the number of selected predictors.

From examining the results, we conclude that priority boosting can be regarded as a practical method that builds interpretable and relatively sparser prediction models. The resulting models are built incorporating prior knowledge and/or practical constraints via its hierarchical structure, which may affect on the prediction ability positively and negatively. Interpretability of the resulting models and implicit variable selection during model estimation are the characteristics that are inherited from boosting.

LBboost is the boosting algorithm with the subset-updating approach that is based on Lasso. It also provides interpretable prediction models and performs automated variable selection. The following findings distinguish this methods from the other methods. LBboost was able to reach as good prediction accuracy as the two priority methods in the AML data, where glmboost and Lasso seems

to have been overfitting. It is presumed that the subset-updating approach in LBboost helped to select relevant predictors that explain the variability in the responses, and resulting models were able to avoid overfitting. It should be noted that LBboost did not use the prior knowledge unlike the priority methods, but processed data purely data-driven.

Also in the simulation study, LBboost often showed relatively good results, both in the scenarios where the naive methods, Lasso and glmboost, gave larger errors and in the scenarios where the priority methods gave larger errors. When it comes to the number of selected predictors, LBboost often chose the largest or the second largest number of predictors of the five methods, in contrast to the priority methods' property of providing sparser models.

In summary, LBboost is a method that derives prediction models purely data-driven, which can reach similar or better prediction accuracy compared to the priority methods in our data. Further, the results suggests that LBboost works well in the situations where the naive methods are overfitting.

6.2 Future work

Our findings presented the similarity of the properties of priority boosting and priority-Lasso, while it remains to discuss the differences between the two methods. One characteristic of boosting algorithms is their stability in the high-dimensional data. It is known that Lasso tends to suffer from the multi-collinearity problem (Hastie & Tibshirani, 1990). In the situation where there are correlations among blocks of the predictors, the coefficients of the predictors selected by Lasso fluctuate widely as the strength of regulation varies. On the other hand, boosting's regularization paths tend to monotone. It would be interesting to compare priority boosting and priority-Lasso under different correlation structures, to see if they inherit the characteristics of their "base" methods.

Another boosting characteristic is flexibility derived from its modular nature, i.e., boostings can basically combine any type of base learners and loss functions. This thesis focus on the priority boosting algorithm defined based on `glmboost` for simplicity. However, it is possible to consider priority boosting that is built based on other boosting methods. For example, if we use `gamboost` instead of `glmboost`, then it allows the resulting models to explain nonlinear effects of predictors on the responses. It makes a difference from the prediction ability of priority-Lasso.

In the appendix A, we present the R functions `prioritybоost`. It is presumed that the setting in the option `cvoffset` in `prioritybоost` may have an influence on prediction accuracy. As discussed in Section 2.4.3, with their hierarchical structure, the resulting model of the priority methods tends to underestimate the influences of blocks with lower priorities, which might make the prediction accuracy worse. `cvoffset` is an option to address this problem by implementing cross-validated offsets, which was devised by Klau et al. (2018). (This option can be found also in function `prioritylasso`). In our simulation study and the application to the AML data, we did not use the version with cross-validated offsets because our aim is to see the impact of the hierarchical

structure clearly and using this version increases computational time. However, it also makes sense to evaluate the prediction accuracy by this version.

Finally, the AML data is an illustrative example to see the properties of priority boosting and LBboost on a real multi-omics dataset. In order to get a solid answer on the evaluation of their prediction abilities on multi-omics data, a large number of data sets need to be investigated. To achieve this purpose, cancer datasets from the database 'The Cancer Genome Atlas' (TCGA) may be used.

Appendices

APPENDIX A

R implementations for priority boosting and LBboost

We present the R implementations for priority boosting and Lasso-based block boosting (LBboost). Appendix A.1 displays function `prioritybоost`, which provides the coefficient estimates by priority boosting for continuous responses and for time-to-event responses, and its supplementary function `makeCVdivision`. Function `makeCVdivision` can be find in the package prioritylasso, which was developed by Klau et al. (2020). Appendix A.2 shows four functions that implemented LBboost. Function `LBboost` returns the coefficient estimates for continuous responses and for time-to-event responses. This requires a supplementary functions `loss_offset`.

A.1 prioritybоost

```
1  prioritybоost <- function(x, y, blocks, family, cvoffset=FALSE,
   ↪  cvoffsetnfolds=5, nodes){
2    # x: design matrix.
3    # y: vector of responses.
4    # blocks: list showing block structure in the format
     ↪  {list(b1=...,b2=...,)}, where the dots represents the
     ↪  indices of the predictors in this block.
5    # family: "gaussian" for continuous y, "cox" for y of type
     ↪  Surv.
6    # cvoffset: logical, whether the cross-validated offsets
     ↪  should be used. Default is FALSE.
7    # cvoffsetnfolds: the number of folds in the CV procedure
     ↪  for the cross-validated offset.  Default is 5.
8    # nodes: the number of nodes used for parallel computing.
9
10   cl <- makeCluster(nodes)
11   coeff <- list()
12   mb.fit <- list()
13   offlist <- list(NULL) # list of offsets
14   for(i in 1:length(blocks)){
15     # Create matrix for the target block
```

```
16      ind_block <- blocks[[i]]
17      x_block <-  x[,ind_block]
18
19      # Fit glmboost with offset
20      if(family == "gaussian") {
21        fit <- glmboost(x=x_block, y=y, family=Gaussian(),
          ↪  offset = offlist[[i]], control=boost_control(mstop =
          ↪  300, nu = 0.1))
22        myApply <- function(dataset, glmboost ,...) {
23          myFun <- function(...) {
24            library("mboost")
25            glmboost(...)
26          }
27          parLapply(cl=cl, dataset, myFun, ...)
28        }
29        cvm <- cvrisk(fit, papply = myApply)
30      }
31
32      if(family == "cox") {
33        fit <- glmboost(x=x_block, y=y, family=CoxPH(), offset =
          ↪  offlist[[i]],
34        control=boost_control(mstop = 300, nu = 0.1))
35        myApply <- function(dataset, glmboost ,...) {
36          myFun <- function(...) {
37            library("mboost")
38            glmboost(...)
39          }
40          parLapply(cl=cl, dataset, myFun, ...)
41        }
42        cvm <- cvrisk(fit, papply = myApply)
43      }
44
45      mb.fit[[i]] <- fit[mstop(cvm)]
46
47      # Compute offset with CV for the next block
48      if(cvoffset) {
49        cvdiv <- makeCVdivision(n = nrow(x), K = cvoffsetnfolds,
          ↪  nrep = 1)[[1]]
50        pred <- matrix(nrow = nrow(x), ncol = 1)
51        for(k in 1:length(cvdiv)) {
52          # For the first block
53          if(is.null(offlist[[i]])){
54            if(family == "gaussian") {
55              fittemp <- glmboost(y=y[cvdiv[[k]]==1,],
                ↪  x=x_block[cvdiv[[k]]==1,], offset = NULL,
                ↪  family = Gaussian(),
                ↪  control=boost_control(mstop = 300, nu=0.1))
56            }
57            if(family == "cox") {
```

```
58            fittemp <- glmboost(y=y[cvdiv[[k]]==1,],
              ↪  x=x_block[cvdiv[[k]]==1,], offset = NULL,
              ↪  family = CoxPH(), control=boost_control(mstop
              ↪  = 300, nu=0.1))
59          }
60
61          fittemp <- fittemp[mstop(cvm)]
62          coe <- coef(fittemp)
63
64          xx <-  as.matrix(x_block)[cvdiv[[k]]==0,
65          names(coe[names(coe) != "(Intercept)"]), drop=FALSE]
66
67          if(length(coe)==0) {
68            lp <- rep(0, nrow(xx))
69          }else{
70            Intercept <- rep(1, nrow(xx))
71            xtemp <-  t(rbind(Intercept,t(xx)))
72            colnames(xtemp)[1] <- "(Intercept)"
73            lp <- as.vector(xtemp[ ,names(coe), drop=FALSE]
              ↪  %*% coe)
74          }
75
76          pred[cvdiv[[k]] == 0,] <-  lp
77        } else {  # For the second block to the last block
78          if(family == "gaussian") {
79            fittemp <- glmboost(y=y[cvdiv[[k]]==1,],
              ↪  x=x_block[cvdiv[[k]]==1,], offset =
              ↪  offlist[[i]][cvdiv[[k]]==1,], family =
              ↪  Gaussian(), control=boost_control(mstop = 300,
              ↪  nu=0.1))
80          }
81          if(family == "cox") {
82            fittemp <- glmboost(y=y[cvdiv[[k]]==1,],
              ↪  x=x_block[cvdiv[[k]]==1,], offset =
              ↪  offlist[[i]][cvdiv[[k]]==1,], family = CoxPH(),
              ↪  control=boost_control(mstop = 300, nu=0.1))
83          }
84          fittemp <- fittemp[mstop(cvm)]
85
86          coe <- coef(fittemp)
87          xx <-  as.matrix(data_block)[cvdiv[[k]]==0,
88          names(coe[names(coe) != "(Intercept)"]), drop=FALSE]
89
90          if(length(coe)==0) {
91            lp <- rep(0, nrow(xx))
92          }else{
93            Intercept <- rep(1, nrow(xx))
94            xtemp <-  t(rbind(Intercept,t(xx)))
95            colnames(xtemp)[1] <- "(Intercept)"
```

```
96              lp <- as.vector(xtemp[ ,names(coe), drop=FALSE]
                ↪  %*% coe)
97            }
98
99            pred[cvdiv[[k]] == 0,] <-
              ↪  offlist[[i]][cvdiv[[k]]==0] + lp
100         }
101       }
102     }
103     else{  # Compute offset without CV
104       pred <- predict(mb.fit[[i]], type="link")
105     }
106
107     offlist[[i+1]] <- as.matrix(pred)
108
109     coe <- coef(mb.fit[[i]])
110     coeff[[i]] <- coe[names(coe) != "(Intercept)"]
111   }
112   stopCluster(cl)
113   coeff[[i]] <- coe
114   finallist <- list(coefficients = unlist(coeff), mboost.fit =
      ↪  mb.fit, call = match.call())
115   return(finallist)
116 }
```

```
1  makeCVdivision <- function(n=300, K=5, nrep=1) {
2    ngroup <- rep(floor(n/K), times=K)
3    if(n - sum(ngroup) != 0)
4      ngroup[1:(n - sum(ngroup))] <- ngroup[1:(n - sum(ngroup))]
       ↪  + 1
5
6    cvlist <- list()
7
8    for(i in 1:nrep) {
9      indgroup <- sample(rep(1:K, times=ngroup))
10     cvlist[[i]] <- list()
11     for(j in 1:K) {
12       tempvec <- rep(0, n)
13       tempvec[indgroup!=j] <- 1
14       cvlist[[i]][[j]] <- tempvec
15     }
16   }
17   return(cvlist)
18 }
```

A.2 LBboost

```
LBboost <- function(x, y, blocks, family, nu=0.1, itermax,
no_update_max=20, cv_frequency=1){
  # x: design matrix
  # y: vector for response variables
  # blocks: list showing block structure in the format
  ↪ {list(b1=...,b2=...,)}, where the dots represents the
  ↪ indices of the predictors in this block.
  # family: "gaussian" or "cox"
  # nu: step length parameter
  # itermax: the number of iterations to perform
  # no_update_max: When the number of times LBboost does not
  ↪ update the coefficients reaches no_update_max, it stops
  ↪ iteration and returns the estimates.
  # cv_frequency: parameter that specifies how often cv.glmnet
  ↪ is used.

  # Set zero vector as initial values of coefficients for all
  ↪ predictors
  coeall = rep(0, ncol(x))
  names(coeall) <- colnames(x)
  offlist <- list(NULL)  # list of offsets for each iteration
  coelist <-list()       # list of coefficients for each
  ↪ iteration
  cvmvec <- vector()     # vector of minimum cvm that gmlnet
  ↪ computes in each iteration
  loss_train <- vector() # vector of MSE or negative log
  ↪ likelihood in each iteration with training set
  loss_valid <- vector() # vector of MSE or negative log
  ↪ likelihood in each iteration with validation set
  count_no_update <- 0   # Counter for no update (adding 1
  ↪ when cv.glmnet returns NULL models for all blocks in the
  ↪ actual iteration)
  no_coeff_count <- 0
  lambda_temp1 <- vector()
  lambda_temp2 <- vector()
  lambda_list <- list(b1=NULL, b2=NULL, b3=NULL, b4=NULL)

  itr <- 1
  while(itr <= itermax){
    loss_itr <- vector()   # vector of MSE or deviance with
    ↪ each candidate model in the actual iteration
    coelist_itr <- list()  # list of coefficients with each
    ↪ candidate model in the actual iteration

    if(itr%%cv_frequency==0 | itr==1 | itr==2){
      for(i in 1:length(blocks)){
        # model matrix for the target block
```

```
34          block <- x[,blocks[[i]]]
35          # Fit lasso with offset
36          if(family=="gaussian"){
37            fitlasso <- cv.glmnet(block, y, family = "gaussian",
              ↪  offset = offlist[[itr]], type.measure = "mse",
              ↪  standardize = TRUE, parallel = TRUE)
38          }
39          if(family=="cox"){
40            fitlasso <- cv.glmnet(block, y, family = "cox",
              ↪  offset = offlist[[itr]], type.measure =
              ↪  "deviance", standardize = TRUE, parallel = TRUE)
41          }
42          if(itr==1){
43            lambda_temp1[i] <- fitlasso$lambda.min
44            lambda_list[[i]][[1]] <- fitlasso$lambda.min
45          }else if(itr==2){
46            lambda_temp2[i] <- fitlasso$lambda.min
47            lambda_list[[i]][[2]] <- fitlasso$lambda.min
48          }else {
49            lambda_temp1[i] <- lambda_temp2[i]
50            lambda_temp2[i] <- fitlasso$lambda.min
51            lambda_list[[i]][[length(lambda_list[[i]])+1]] <-
              ↪  fitlasso$lambda.min
52          }
53
54          # coefficients chosen from the target block
55          coe <- fitlasso$glmnet.fit$beta[,fitlasso$lambda ==
            ↪  fitlasso$lambda.min]
56          coelist_itr[i] <- list(coe[coe!= 0])
57
58          loss_itr[i] <- min(fitlasso$cvm)
59        }
60      }else{
61        for(i in 1:length(blocks)){
62          # model matrix for the target block
63          block <- x[,blocks[[i]]]
64
65          # Make lambda sequence
66          lambda <- c(lambda_temp1[i], lambda_temp2[i])
67          a <- lambda_temp1[i]/1
68          b <- lambda_temp2[i]/1
69          lambda <- c(a, b)
70
71          # Fit lasso with offset
72          if(family=="gaussian"){
73            fitlasso <- glmnet(block, y, family = "gaussian",
              ↪  offset = offlist[[itr]], type.measure = "mse",
              ↪  lambda=lambda, standardize = TRUE, parallel =
              ↪  TRUE)
74          }
```

```
75          if(family=="cox"){
76            fitlasso <- glmnet(block, y, family = "cox", offset
              ↪  = offlist[[itr]], type.measure = "deviance",
              ↪  lambda=lambda, standardize = TRUE, parallel =
              ↪  TRUE)
77          }
78
79          # coefficients chosen from the target block
80          coe <- fitlasso$beta[,"s1"]
81          coelist_itr[i] <- list(coe[coe!= 0])
82          loss_itr[i] <- loss_offset(fitlasso, coelist_itr[[i]],
             ↪  offlist[[itr]], x, y)
83        }
84      }
85
86      # Update coefficients
87      ind <- which.min(loss_itr)
88      cvmvec[itr] <- loss_itr[ind]
89      coe <- coelist_itr[[ind]]
90       # If learners chose no coefficient in the current
          ↪  iteration
91      if(length(coe) == 0) {
92        print("No update")
93        count_no_update <- count_no_update + 1
94        if (count_no_update==no_update_max) {
95          print("No updates has been accumulated")
96          break   # Stop iterations
97        }
98        if(itr!=1){
99          offlist[itr+1] <- list(offlist[[itr]])
100         coelist[itr] <- coelist[itr-1]
101       }else{
102         print("Chosen model has no coefficient.")
103         no_coeff_count <- no_coeff_count + 1
104         if(no_coeff_count==3){
105           print("LBboost chose no coefficient.")
106           finallist <- list(coefficients = coeall)
107           return(finallist)
108         }
109         next
110       }
111     }else{      # If leaner(s) chose coefficient(s) in the
          ↪  current iteration
112       for(j in 1:length(coe)){
113         coeall[names(coeall) == names(coe[j])] <-
            ↪  coeall[names(coeall) == names(coe[j])]+ nu*coe[j]
114       }
115       coe_nonzero <- coeall[coeall!=0]
116
```

```
      offlist[itr+1] <- list(as.matrix(x[,names(coe_nonzero),
      ↪  drop=FALSE] %*% coe_nonzero))
      coelist[itr] <- list(coe_nonzero)
    }
    itr <- itr + 1
  }
  finallist <- list(lam=lambda_list, coefficients=coe_nonzero,
  ↪  coefflist=coelist, iter=itr, cvm=cvmvec,
  ↪  call=match.call())
  return(finallist)
}
```

```
loss_offset <- function(object, coe, offset=NULL, new_x,
↪  new_y){
  if(object$call$family == "gaussian"){
    # Compute fitted linear predictor and MSE
    if(length(coe)!=0) {
      Intercept <- rep(1, nrow(new_x))
      xtemp <-  t(rbind(Intercept,t(new_x)))
      colnames(xtemp)[1] <- "(Intercept)"
      lp <- as.vector(xtemp[ ,names(coe), drop=FALSE] %*% coe)
      ↪  + offset
      loss <- sum((new_y - lp)^2)/ length(new_y)  # MSE
    } else{
      lp <- offset
      loss <- sum((new_y - lp)^2)/ length(new_y)
    }
  }
  if(object$call$family == "cox"){
    # Compute fitted linear predictor and negative partial log
    ↪  likelihood
    if(length(coe)!=0) {
      Intercept <- rep(1, nrow(new_x))
      xtemp <-  t(rbind(Intercept,t(new_x)))
      colnames(xtemp)[1] <- "(Intercept)"
      lp <- as.vector(xtemp[ ,names(coe), drop=FALSE] %*% coe)
      ↪  + offset
      dt <- as.data.frame(cbind(new_y, lp))
      fit <- coxph(Surv(time, status)~lp, data=dt, x=TRUE)
      loss <- (-fit$loglik[2])  #  negative partial log
      ↪  likelihood
    } else{
      dt <- as.data.frame(as.matrix(new_y))
      lp <- offset
      fit <- coxph(Surv(time, status)~offset, data=dt, x=TRUE)
      loss <- (-fit$loglik[1])
    }
  }
```

```
32   return(loss)
33 }
```

APPENDIX B

R codes to generate simulation data

We present R codes to generate the data in our simulation study in Chapter 4. The codes for Gaussian responses are in Appendix B.1 and for survival times in Appendix B.2.

B.1 Simulation data for Gaussian responses

```r
###### parameters and blocks for the LLLH setting
# beta
beta_b1 <- c(rep(0.9, 5), rep(0, 5))  #10
beta_b2 <- c(rep(0.7, 5), rep(0, 5))  #10
beta_b3 <- c(rep(0.3, 5), rep(0, 5))  #10
beta_b4 <- c(rep(0.1, 5), rep(0, 995)) #1000
beta <- c(beta_b1, beta_b2, beta_b3, beta_b4)

# blocks
blocks <- list(b1=1:10, b2=11:20, b3=21:30, b4=31:1030)
n1 <- paste("A", 1:10, sep = "")      # 10
n2 <- paste("B", 11:20, sep = "")     # 10
n3 <- paste("C", 21:30, sep = "")     # 10
n4 <- paste("D", 31:1030, sep = "")   # 1000


###### parameters and blocks for the LHHH setting
# beta
beta_b1 <- c(rep(0.9, 5), rep(0, 5))  #10
beta_b2 <- c(rep(0.7, 5), rep(0, 995))  #1000
beta_b3 <- c(rep(0.3, 5), rep(0, 995))  #1000
beta_b4 <- c(rep(0.1, 5), rep(0, 995))  #1000
beta <- c(beta_b1, beta_b2, beta_b3, beta_b4)

# blocks
blocks <- list(b1=1:10, b2=11:1010, b3=1011:2010,
  ↪  b4=2011:3010)
n1 <- paste("A", 1:10, sep = "")        # 10
```

```
n2 <- paste("B", 11:1010, sep = "")     # 1000
n3 <- paste("C", 1011:2010, sep = "")   # 1000
n4 <- paste("D", 2011:3010, sep = "")   # 1000


##### the number of individuals and predictors
n <- 300
p <- length(beta)
```

```
simgaussDepIng <- function(n=300, d=0.9, beta, blocks){
# Generate data for Gaussian responses where there are
  ↪ dependencies within each group
  x <- NULL
  for (i in 1:length(blocks)){
    p <- length(blocks[[i]])
    mu <- rep(0, p)
    I <- diag(p)
    sgm <- d^abs(row(I)-col(I))
    x_block <- mvrnorm(n, mu, sgm)
    x <- cbind(x, x_block)
  }
  x <- scale(x)
  y <- x%*%beta + matrix(rnorm(n), nrow=n)
  return(list(x,y))
}


simgaussDepAmg <- function(n=300, d=0.9, beta, blocks){
# Generate data for Gaussian responses where there are
  ↪ dependencies among groups
  p <- length(beta)
  I <- diag(p)
  sgm <- d^abs(row(I)-col(I))
  mu <- rep(0, p)
  x <- mvrnorm(n, mu, sgm)
  x <- scale(x)
  y <- x%*%beta + matrix(rnorm(n), nrow=n)
  return(list(x,y))
}
```

These functions require package MASS.

```
# Simulate independent data
X <- matrix(rnorm(n*p),n,p)
X <- scale(X)
colnames(X) <- c(n1,n2,n3,n4)
Y <- X%*%beta + matrix(rnorm(n),n,1)

```

```

# Simulate data where there are dependencies within each group
dt <- simgaussDepIng(n=300, d=0.9, beta, blocks)
X <- dt[[1]]
Y <- dt[[2]]
colnames(X) <- c(n1,n2,n3,n4)


# Simulate data where there are dependencies among groups
dt <- simgaussDepAmg(n=300, d=0.9, beta, blocks)
X <- dt[[1]]
Y <- dt[[2]]
colnames(X) <- c(n1,n2,n3,n4)
```

B.2 Simulation data for survival times

```
###### parameters and blocks for the LLLH setting
# beta
beta_b1 <- c(rep(0.9, 5), rep(0, 5))  #10
beta_b2 <- c(rep(0.7, 5), rep(0, 5))  #10
beta_b3 <- c(rep(0.3, 5), rep(0, 5))  #10
beta_b4 <- c(rep(0.1, 5), rep(0, 995)) #1000
beta <- c(beta_b1, beta_b2, beta_b3, beta_b4)

# blocks
blocks <- list(b1=1:10, b2=11:20, b3=21:30, b4=31:1030)
n1 <- paste("A", 1:10, sep = "")      # 10
n2 <- paste("B", 11:20, sep = "")     # 10
n3 <- paste("C", 21:30, sep = "")     # 10
n4 <- paste("D", 31:1030, sep = "")   # 1000


###### parameters and blocks for the LHHH setting
# beta
beta_b1 <- c(rep(0.9, 5), rep(0, 5))  #10
beta_b2 <- c(rep(0.7, 5), rep(0, 995))  #1000
beta_b3 <- c(rep(0.3, 5), rep(0, 995))  #1000
beta_b4 <- c(rep(0.1, 5), rep(0, 995))  #1000
beta <- c(beta_b1, beta_b2, beta_b3, beta_b4)

# blocks
blocks <- list(b1=1:10, b2=11:1010, b3=1011:2010,
  ↪  b4=2011:3010)
n1 <- paste("A", 1:10, sep = "")        # 10
n2 <- paste("B", 11:1010, sep = "")     # 1000
n3 <- paste("C", 1011:2010, sep = "")   # 1000
n4 <- paste("D", 2011:3010, sep = "")   # 1000

```

```r

##### the number of individuals and predictors
n <- 300
p <- length(beta)
```

```r
simcoxIndep <- function(n=300, beta){
# Generate data for survival times where all variables are independent
  p <- length(beta)
  x <- matrix(rnorm(n*p), n, p)
  x <- scale(x)
  lp <- x%*%beta
  u <- runif(n)
  a=1; b=0.01  #parameter for Weibull distribution
  lifetime <- (- log(u) / (b * exp(lp)))^(1 / a)/100

  # censoring time
  cen_time <- rexp(n, 0.3)

  # observed time
  time <- pmin(lifetime, cen_time)
  status <-  as.numeric(lifetime <= cen_time )

  dat <-  data.frame(time, status, x)
  return(dat)
}


simcoxDepIng <- function(n=300, d=0.9, beta, blocks){
# Generate data for survival times where there are dependencies within each group
  x <- NULL
  for (i in 1:length(blocks)){
    p <- length(blocks[[i]])
    mu <- rep(0, p)
    I <- diag(p)
    sgm <- d^abs(row(I)-col(I))
    x_block <- mvrnorm(n, mu, sgm)
    x <- cbind(x, x_block)
  }
  x <- scale(x)

  lp <- x%*%beta
  u <- runif(n)
  a=1; b=0.01    # parameters for Weibull distribution
  lifetime <- (- log(u) / (b * exp(lp)))^(1 / a)/100

  # censoring time
  cen_time <- rexp(n, 0.3)
```

```r

  # observed time
  time <- pmin(lifetime, cen_time)
  status <-  as.numeric(lifetime <= cen_time )

  dat <-  data.frame(time, status, x)
  return(dat)
}


simcoxDepAmg <- function(n=300, d=0.9, beta, blocks){
# Generate data for survival times where there are
↪  dependencies among groups
  p <- length(beta)
  I <- diag(p)
  sgm <- d^abs(row(I)-col(I))
  mu <- rep(0, p)
  x <- mvrnorm(n, mu, sgm)
  x <- scale(x)

  lp <- x%*%beta
  u <- runif(n)
  a=1; b=0.01   # parameters for weibull distribution
  lifetime <- (- log(u) / (b * exp(lp)))^(1/a)/100

  # censoring time
  cen_time <- rexp(n, 0.3)

  # observed time
  time <- pmin(lifetime, cen_time)
  status <-  as.numeric(lifetime <= cen_time )

  dat <-  data.frame(time, status, x)
  return(dat)
}
```

Function `simcoxDepIng` and `simcoxDepAmg` require package MASS.

```r
# Simulate independent data
data <- simcoxIndep(n=n, beta=beta)
Y <- Surv(data$time, data$status)
X <- as.matrix(data[,3:dim(data)[2]])
colnames(X) <- c(n1,n2,n3,n4)


# Simulate data where there are dependencies within each group
data <- simcoxDepIng(n=n, beta=beta, blocks=blocks)
Y <- Surv(data$time, data$status)
X <- as.matrix(data[,3:dim(data)[2]])
```

```
colnames(X) <- c(n1,n2,n3,n4)


# Simulate data where there are dependencies among groups
data <- simcoxDepAmg(n=n, beta=beta, blocks=blocks)
Y <- Surv(data$time, data$status)
X <- as.matrix(data[,3:dim(data)[2]])
colnames(X) <- c(n1,n2,n3,n4)
```

Bibliography

Aalen, O. (1978). Nonparametric inference for a family of counting processes. *The Annals of Statistics* **6**, 701 – 726.

Aalen, O., Borgan, Ø. & Gjessing, H. (2008). *Survival and Event History Analysis: A Process Point of View (Statistics for Biology and Health)*. Springer.

Bender, R., Augustin, T. & Blettner, M. (2005). Generating survival times to simulate cox proportional hazards models. *Statistics in Medicine* **24**.

Bühlmann, P. & Hothorn, T. (2007). Boosting algorithms: Regularization, prediction and model fitting. *Statistical science* **22**, 477–505.

Büchner, T., Berdel, W. E., Schoch, C., Haferlach, T., Serve, H. L., Kienast, J., Schnittger, S., Kern, W., Tchinda, J., Reichle, A., Lengfelder, E., Staib, P., Ludwig, W.-D., Aul, C., Eimermacher, H., Balleisen, L., Sauerland, M.-C., Heinecke, A., Wörmann, B. & Hiddemann, W. (2006). Double induction containing either two courses or one course of high-dose cytarabine plus mitoxantrone and postremission therapy by either autologous stem-cell transplantation or by prolonged maintenance for acute myeloid leukemia. *Journal of Clinical Oncology* **24**, 2480–2489.

Bühlmann, P. & Yu, B. (2003). Boosting with the l_2 loss: Regression and classification. *Journal of the American Statistical Association* **98**, 324–339.

Cox, D. R. (1972). Regression models and life-tables. *Journal of the royal statistical society series b-methodological* **34**, 187–220.

De Bin, R. (2016). Boosting in cox regression: a comparison between the likelihood-based and the model-based approaches with focus on the r-packages coxboost and mboost. *Computational Statistics* **31**, 513–531.

Freund, Y. (1990). Boosting a weak learning algorithm by majority. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, COLT '90. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Freund, Y. & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Machine Learning, Proceedings of the Thirteenth International Conference (ICML '96), Bari, Italy, July 3-6, 1996*, L. Saitta, ed. Morgan Kaufmann.

Friedman, J., Hastie, T. & Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors). *The Annals of Statistics* **28**, 337 – 407.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics* **29**, 1189–1232.

Grimwade, D., Hills, R. K., Moorman, A. V., Walker, H., Chatters, S., Goldstone, A. H., Wheatley, K., Harrison, C. J., Burnett, A. K. & Group, N. C. R. I. A. L. W. (2010). Refinement of cytogenetic classification in acute myeloid leukemia: determination of prognostic significance of rare recurring chromosomal abnormalities among 5876 younger adult patients treated in the united kingdom medical research council trials. *Blood, The Journal of the American Society of Hematology* **116**, 354–365.

Hastie, T. & Tibshirani, R. (1990). *Generalized Additive Models*. Chapman and Hall.

Hastie, T., Tibshirani, R. & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics)*. Springer.

Herrmann, M., Probst, P., Hornung, R., Jurinovic, V. & Boulesteix, A.-L. (2020). Large-scale benchmark study of survival prediction methods using multi-omics data. *Briefings in Bioinformatics* **22**.

Hofner, B., Mayr, A., Robinzonov, N. & Schmid, M. (2014). Model-based boosting in R: A hands-on tutorial using the R package mboost. *Computational Statistics* **29**, 3–35.

Hothorn, T., Buehlmann, P., Kneib, T., Schmid, M. & Hofner, B. (2021). *mboost: Model-Based Boosting*. R package version 2.9-5.

Kaplan, E. L. & Meier, P. (1958). Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association* **53**, 457–481.

Kearns, M. & Valiant, L. G. (1989). Crytographic limitations on learning boolean formulae and finite automata. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC '89. New York, NY, USA: Association for Computing Machinery.

Klau, S., Hornung, R. & Bauer, A. (2020). *prioritylasso: Analyzing Multiple Omics Data with an Offset Approach*. R package version 0.2.5.

Klau, S., Jurinovic, V., Hornung, R., Herold, T. & Boulesteix, A. (2018). Priority-lasso: a simple hierarchical approach to the prediction of clinical outcome using multi-omics data. *BMC Bioinform.* **19**, 322:1–322:14.

Mayr, A., Binder, H., Gefeller, O. & Schmid, M. (2014). The evolution of boosting algorithms from machine learning to statistical modelling. *Methods of information in medicine* **53**.

Nelson, W. (1969). Hazard plotting for incomplete failure data. *Journal of Quality Technology* **1**, 27–52.

Nelson, W. (1972). Theory and applications of hazard plotting for censored failure data. *Technometrics* **14**, 945–966.

Ridgeway, G. K. (1999). *Generalization of boosting algorithms and applications of bayesian inference for massive datasets.* Ph.D. thesis, University of Washington.

Schapire, R. E. (1990). The strength of weak learnability. *Machine learning* **5**, 197–227.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* **58**, 267–288.

Tibshirani, R. (1997). The lasso method for variable selection in the cox model. *Statistics in Medicine* **16**, 385–395.