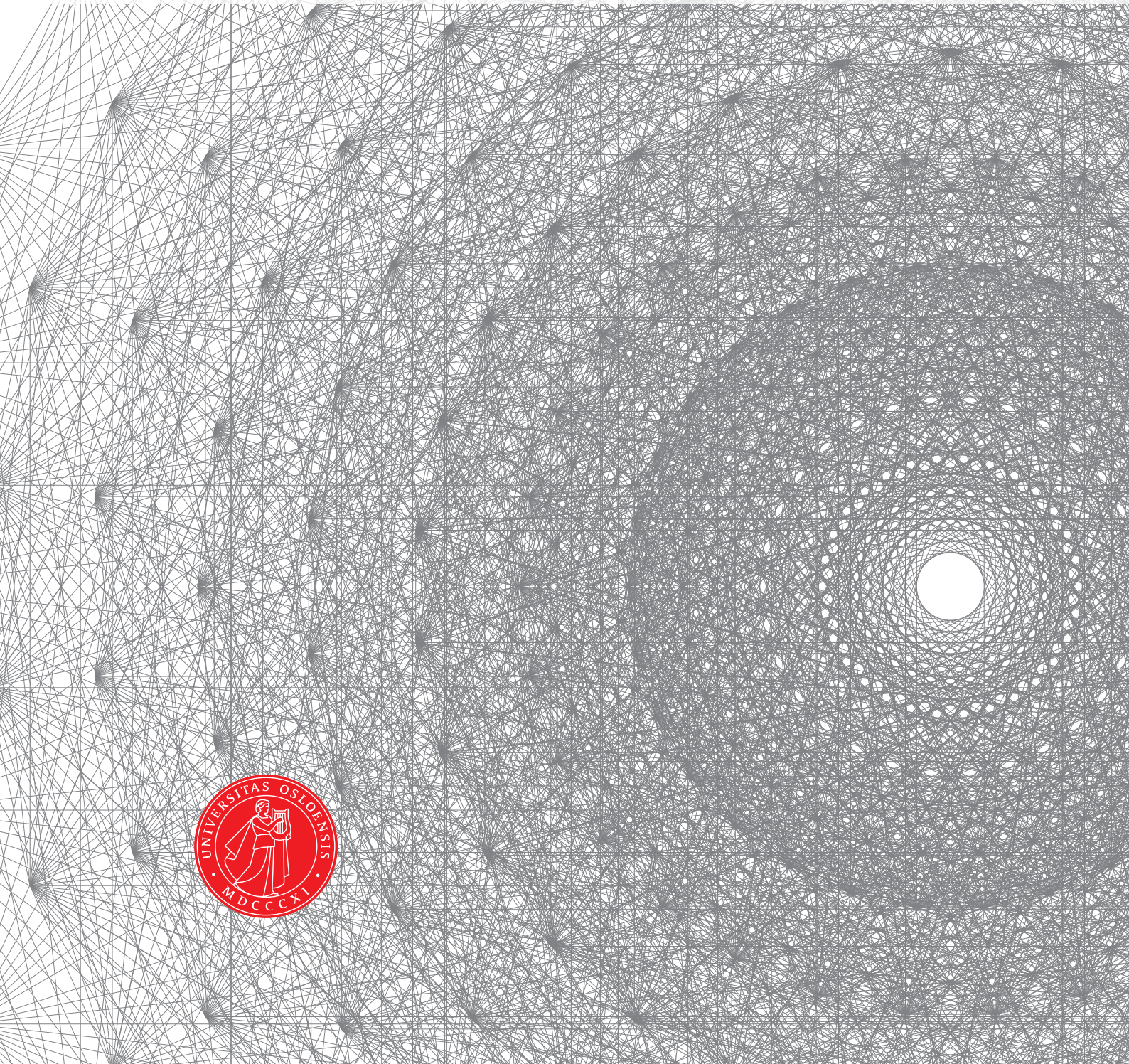
UiO : Department of Mathematics
University of Oslo

A gradient boosting algorithm to extend first-hitting time models to a high-dimensional survival setting

Vegard Stikbakke

Master's Thesis, Spring 2019



This master's thesis is submitted under the master's program *Modelling and Data Analysis*, with program option *Statistics and Data Analysis*, 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

In this thesis, we consider models for survival data with high-dimensional explanatory covariates. Specifically, we consider a typical setting in which both genomic data and clinical data are used to model the survival. In a high-dimensional setting, models which can perform variable selection and shrinkage are needed. Often, the Cox regression model is used. It requires that hazard rates of individuals are proportional. This is hard to verify in high dimensions, and it is hard to justify when doing variable selection. Furthermore, when using both clinical and genomic data it is difficult to combine the data in a sensible way. We therefore discuss first hitting time models, specifically with a Wiener health process. Such models model the health process of each individual, based on an individual initial health level and an individual drift. Within such a model, it is sensible to assign genomic data to the initial level, and the clinical data to the drift of the health process. To estimate parameters in a first hitting time model with a Wiener health process, we implement a gradient boosting algorithm which we call *FHTBoost*. Boosting multivariate distributions is complex, and may be computationally intensive, but based on recent advances, we are able to make do with one tuning parameter. We perform a simulation study where the implemented algorithm manages to achieve good variable selection and model fit. Finally, FHTBoost is applied to a data set of neuroblastoma, where it achieves a predictive power, measured by Brier score, that is comparable to a Cox model.

Contents

Chapter 1

Introduction and outline of the thesis

sec:intro

Survival data are data that is concerned with the time to an event. We typically study a cohort of patients after a cancer surgery, and we record the time to a possible recurrence of the cancer, or we study the time it takes from a child from a set of parents is born until a second child is born. If the event happens, we record the observed time. Not all statistical units experience an event during the time in which the study is conducted. For some of them we only know the last time that is recorded, e.g. at the latest check-up. These observations are right-censored, and add some complications in the data analysis.

Almost all practical modeling of survival data is done with the famous Cox regression model (Cox, 1972). One of its characteristics is that it models the hazard. The hazard of an individual at a time t is the probability that the individual experiences an event, given that the event has not happened yet, in the infinitesimal time interval at t. A key underlying assumption of the Cox model is that individuals have proportional hazards at all times, i.e. the ratio of their hazards does not change over time. This assumption is not always valid, and there is a need for more flexible models. Among several options, in this thesis we will focus on first hitting time models (FHT) (Lee and Whitmore, 2006). Instead of modeling the hazard, the main idea of FHT models is to model the underlying process of an individual before an event occurs. To model its life, so to speak. The FHT model framework is a rich and flexible modeling framework, where we specify a stochastic process and an appropriate threshold, barrier, or boundary, depending on the association we wish to evoke. When the process hits the barrier, or crosses the boundary, the event is triggered. The time to event is therefore the time that passes from the process starts until it hits the barrier. For certain choices of processes, there exist fully parameterized expressions which can be used in regression. This is the case for the Wiener process with initial level and drift, which leads to a bivariate probability distribution. As usual, the parameters of this distribution are related to explanatory variables, in a regression setup with additive predictors.

An important part of modern biomedical statistics is the ability to deal with high-dimensional data, for example genetic data. Genetic data are nowadays widely available, and typical data sets include gene expressions from genes numbering in the tens of thousands. Such data are called high-dimensional data,

because there are usually more variables than observations. A scenario where the number of covariates p, is much larger than the number of predictors, n, also often referred to as the $p \gg n$ scenario. One can think of the genes from one individual as one point in a high-dimensional space where each gene spans one dimension. Somewhat counterintuitively, virtually all points in high-dimensional space will be far apart. Especially if there are few points in this space compared with the number of dimensions of the space, it makes it very easy for statistical models to overfit, i.e. to explain the variation in the data in a way that is not really true. Furthermore, estimated models that are overfit would not carry over predictive power to unseen data of a similar kind. Many classical statistical models are simply unable to use so many predictors, at least directly.

When it comes to survival analysis, there are models which extend Cox regression to such settings. They perform empirically well, but it is somewhat unclear how the check the proportional hazards assumption. In contrast, there do not exist similar extensions for the FHT model. To the best of our knowledge, there has been no attempt at developing methods for first hitting time regression in a high-dimensional setting. In this thesis we try to fill this gap by developing a boosting algoritm which allows fitting a FHT model with high-dimensional data. Gradient boosting is an algorithmical framework that has been very successful in high-dimensional ($p \gg n$) scenarios.

It originated at the end of the 20th century in the field of machine learning. Friedman (2001) later provided a statistical view. Gradient boosting algorithms are iterative algorithms for performing regularized minimization of a loss function, which can very well be a negative log likelihood function. Gradient boosting is still very much an active field of research, and various methods exist for model fitting. Most traditional gradient boosting algorithms are concerned with modelling one parameter. The Wiener FHT model we use is a bivariate model, and we can therefore not use the lasso, nor can we use the standard gradient boosting methods. In more recent years, however, gradient boosting methods for regression of parameters beyond the mean have also been introduced.

In Chapter 2, we discuss the survival analysis setting, Cox regression, and FHT models. These are first discussed in general, before addressing in detail the specific case where we use a Wiener process. In Chapter 3, we discuss gradient boosting, including recent methods for estimating multidimensional models (Thomas et al., 2018). The main result of this thesis can be found in Chapter 4, where we develop a multidimensional, component-wise gradient boosting algorithm to fit the bivariate FHT model with Wiener processes, which we call *FHTBoost*. We implement this boosting algorithm entirely from scratch. Using *FHTBoost*, we can estimate the linear additive predictors of the FHT model.

Chapter 5 explains evaluation measures that we then use in subsequent Chapters. In Chapter 6, we perform a simulation study where we verify that *FHTBoost* manages to estimate the parameters of the FHT model. We consider two scenarios, one with uncorrelated data, and a more realistic scenario with highly correlated data. Chapter 7 looks at a survival data set consisting children diagnosed with neuroblastoma. We estimate an FHT model on this data, and discuss the results. We then compare its predictive performance with that of a boosted Cox regression. Finally, we provide a conclusion and some ideas for future work.

Chapter 2

First-hitting-time models

2.1 Survival data

Time to event data are seen in many different contexts, including medicine, engineering, biology, and sociology. One considers a set of individuals i, $i = 1, 2, \ldots, N$, for which an event can happen. The term *survival data* is used to refer to time to event data where such events only can happen once. An overview of modelling of survival data can be found in for example Aalen et al. (2008). Althought the term survival data sounds like it refers to deaths only, the event in question may be anything of interest. You might, for example, be a doctor performing a study of cancer patients, and monitoring them for possible relapse. In this case, the event is the relapse. Or, possibly, you are a demographer looking at all parents who have only one child, and you are monitoring the time that elapses before they have a second child. Clearly, to observe such data in real life, we must wait until the event actually happens. This might in some cases never happen, or it might take a very long time. Consider, for example, a clinical trial of N patients who have been treated for some disease, and where T_i, $i = 1, \ldots, N$, is the time until their relapse. Such a trial can only last a certain amount of time, say, until a time τ. Luckily, not every patient relapses during that time, and so the actual time to their relapse, $\tilde{T}_i$, is not observed. We could throw away these observations without an observed event, and consider them irrelevant. But we at least know that these patients survived until time τ. We therefore work with the concept of incomplete data, which we call *censored* survival data (Aalen et al., 2008).

2.1.1 Independently censored survival data

subsec:survdata

The time-to-event $\tilde{T}$ is a random variable of a non-negative domain. This time-to-event has a corresponding random variable W which represents the censoring time of the time-to-event. The observed, and possibly censored, survival time is

$$T = \min(\tilde{T}, W).$$

We also operate with a corresponding censoring indicator

$$D = I(\tilde{T} = T),$$

which is 1 if the observed survival time is not censored, and 0 if it is. In the clinical trial example mentioned, the censoring time W would be the end of

the possible observation period of the trial, namely τ. An important property of survival data is the concept of *independent censoring*, also called *random censoring*. We say that we have independent censoring if the censoring indicator D is independent of the time, meaning that

$$P(T|D) = P(T).$$

What we have so far called censored data is in truth *right-censored* survival data. Left-censoring is also possible, but we will not discuss it in this thesis, and so we continue to simply use the term "censoring."

2.1.2 The survival function $S(t)$, the hazard function $\alpha(t)$, and their estimators

When studying censored survival data, we are interested in estimating the probability of surviving until time t, which is called the survival function, and is defined as

$$S(t) = \Pr(T > t) = 1 - \Pr(T < t) = 1 - F(t).$$

Here $F(t)$ is the familiar cumulative distribution function. If the derivative of $F(t)$ exists, we denote it $f(t)$, and then the lifetime T has probability distribution function (pdf) $f(t)$.

Another function we are interested in is the hazard function. This is the probability of the event happening at time t, conditioned on the event not having happened yet. More formally, the hazard function is defined as

$$\alpha(t) = \lim_{\epsilon \to 0} \frac{\Pr(T < t + \epsilon | T > t)}{\epsilon}.$$

Estimating the hazard function is hard, and we do not achieve the usual $\sqrt{n}$ convergence. It is, however, typically the function we are most interested in, as we will see later, in subsection 2.2.2. Note that by observing

$$\Pr(T < t + \epsilon | T > t) = \frac{\Pr(T < t + \epsilon, T > t)}{\Pr(T > t)} = \frac{F(t + \epsilon) - F(t)}{S(t)},$$

and inserting this into the hazard function we have the relation

$$\alpha(t) = \frac{1}{S(t)} \lim_{\epsilon \to 0} \frac{F(t + \epsilon) - F(t)}{\epsilon} = \frac{f(t)}{S(t)} = \frac{-S'(t)}{S(t)}, \tag{2.1}$$

{eq:hfs}

where the probability distribution function $f(t)$ is obtained by its limit definition, and we note that $S'(t)$ is the derivative of $1 - F(t)$, which is $-f(t)$. We further note that by interchanging the notation for derivation, we obtain

$$\alpha(t) = \frac{S'(t)}{S(t)} = \frac{\frac{\mathrm{d}S}{\mathrm{d}t}}{S(t)}. \tag{2.2}$$

{eq:alpha-diff}

By integrating the hazard from 0 to time t, we get the cumulative hazard function,

$$A(t) = \int_0^t \alpha(s)\ \mathrm{d}s, \tag{2.3}$$

{eq:cumulative-hazard-1}

and we insert (2.2) into (2.3),

$$A(t) = -\int_0^t \frac{\frac{\mathrm{d}S}{\mathrm{d}s}}{S(s)}\,\mathrm{d}s = -\int_0^t \frac{1}{S(s)}\,\mathrm{d}S = -\log(S(t)), \tag{2.4}$$

{eq:cumulative-hazard}

which is an important relationship. Given survival data $(t_i, d_i)_{i=1}^N$, we introduce the *risk set* $R(t)$, which gives the set of all individuals at risk at time t,

$$R(t) = \{i \colon t_i \geq t\}.$$

We further introduce the function $Y(t)$, which is equal to the number of individuals still at risk at time t,

$$Y(t) = \#R(t) = \#\{i \colon t_i \geq t\},$$

where $\#(\cdot)$ is the counting operator over a set. Note that $Y(t)$ does not depend on the censoring indicators, since an individual is at risk at time t even though it turns out that it did not die, i.e., its censoring indicator d_i is 0. Estimating the survival function $S(t)$ is usually done by the Kaplan-Meier estimator (Kaplan and Meier, 1958),

$$\hat{S}_{\mathrm{KM}}(t) = \prod_{i:\{t_i \leq t\}} 1 - \frac{d_i}{Y(t_i)},$$

and the cumulative hazard function $A(t)$ by the Nelson-Aalen estimator (Nelson, 1972; Aalen, 1978),

$$\hat{A}(t) = \sum_{i:\{t_i \leq t\}} \frac{d_i}{Y(t_i)}.$$

2.2 Classical inference approaches

2.2.1 Likelihood regression

Consider survival data (t_i, d_i), $i = 1, 2, \ldots, N$, where t_i is the time to event of individual i, and d_i is a censoring indicator of the usual type, meaning it is 1 if the event has been observed, and 0 if not. To make inference on what affects the time to event, we need to consider covariates. Covariates are information about an individual. In medical applications, typical covariates are age, gender, disease status, as well as clinical measurements. Denote the covariates, i.e., the information, of an individual i by $x_{i,1}, x_{i,2}, \ldots, x_{i,p}$, where p is the total number of pieces of information. We gather these in a vector $\boldsymbol{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,p})$. We may now consider a data set of complete tuples of survival data with covariates,

$$D = (t_i, d_i, \boldsymbol{x}_i)_{i=1}^N. \tag{2.5}$$

{eq:surv-D}

Now, consider a survival time distribution

$$\psi(\boldsymbol{\beta}), \tag{2.6}$$

{eq:surv-time-dist}

which has a vector of regression parameters $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_p)$, with one parameter β_j, $j = 1, 2, \ldots, p$ corresponding to one covariate x_j. This distribution has a parameterized survival function

$$S(t|\boldsymbol{\beta}, \boldsymbol{x})$$

and a parameterized probability distribution function

$$f(t|\boldsymbol{\beta}, \boldsymbol{x}).$$

Given an observed dataset D (2.5), we wish to make inference on which covariates affect the survival time. One way to do this, having assumed a specific survival distribution $\psi(\boldsymbol{\beta})$, is to construct a so-called (joint) likelihood. The likelihood is a function of the parameters $\boldsymbol{\beta}$ in the distribution, given the observed sample. The likelihood is maximized at the parameters which have the highest probability of yielding the observed sample. If all censoring indicators are 1, meaning all observations are actual events, we are in the usual statistical regression landscape. For each observation, its likelihood is the probability of observing its event at t_i given the parameters and the data,

$$f(t_i|\boldsymbol{\beta}, \boldsymbol{x}_i).$$

A typical assumption when setting up a (joint) likelihood is to assume that the conditional distribution of each observation is independent and identically distributed (*iid*), given the data. Hence the joint likelihood is the product of all the single likelihood contributions,

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{N} f(t_i|\boldsymbol{\beta}, \boldsymbol{x}_i).$$

But we need to consider the case of censored survival data, and we wish to set up a joint likelihood for such a data set. Again assume that the observations $D = \{\boldsymbol{x}_i, y_i\}_{i=1}^{N}$ are independent and identically distributed from a survival distribution $\psi(\boldsymbol{\beta})$. In the case of an uncensored observation i, meaning $d_i = 1$, the single contribution to the likelihood is

$$f(t_i|\boldsymbol{\beta}, \boldsymbol{x}_i) \tag{2.7}$$

{eq:f}

to the likelihood. If the event has not occurred, i.e. the observation is censored, d_i is 0. In this case, we do not have the actual lifetime, and so we cannot use the lifetime distribution, but we must instead use the survival distribution. After all, the fact that this individual is alive at time t_i is informative. Therefore this observation contributes

$$S(t_i|\boldsymbol{\beta}, \boldsymbol{x}_i) \tag{2.8}$$

{eq:S}

to the likelihood. Since an observation can only be either censored or not censored at the same time, d_i is either 0 or 1. This means that we can combine expressions (2.7) and (2.8) in a way that a single observation contributes the product

$$f(t_i|\boldsymbol{x}_i)^{d_i} S(t_i|\boldsymbol{x}_i)^{1-d_i}$$

to the likelihood. Since we assume the observations to be independent, the likelihood is, again, the product of the single complete contributions. The complete likelihood becomes

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{N} f(t_i|\boldsymbol{x}_i, \boldsymbol{\beta})^{d_i} S(t_i|\boldsymbol{x}_i, \boldsymbol{\beta})^{1-d_i}. \tag{2.9}$$

{eq:surv-lik}

It is more convenient to work with the log likelihood,

$$\begin{aligned} l(\boldsymbol{\beta}) &= \log L(\boldsymbol{\beta}) \\ &= \sum_{i=1}^{N} \left[d_i \log f(t_i|\boldsymbol{x}_i, \boldsymbol{\beta}) + (1 - d_i) \log S(t_i|\boldsymbol{x}_i, \boldsymbol{\beta}) \right]. \end{aligned} \tag{2.10}$$

{eq:surv-log-lik-1}

Note that since $A(t) = -\log S(t)$ and $f(t) = \alpha(t)S(t)$, see (2.4) and (2.1), respectively, (2.10) further simplifies to

$$l(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[d_i \log \alpha(t_i|\boldsymbol{x}_i, \boldsymbol{\beta}) - A(t_i|\boldsymbol{x}_i, \boldsymbol{\beta}) \right].$$

For any survival distribution $\psi(\boldsymbol{\beta})$ (2.6) for which either form of $l(\boldsymbol{\beta})$ can be calculated, we can perform maximum likelihood estimation of its parameters.

subsec:ph-reg

2.2.2 Proportional hazards regression

In other fields of statistics, we are often most interested in modelling the probability distribution function and the cumulative distribution function. In survival analysis, however, we are typically more interested in modelling the hazard function. In this subsection, we consider the effect of covariates on the hazard function $\alpha(\cdot)$. How may we use a covariate vector $\boldsymbol{x}$ in modelling the hazard rate? A very common model to choose here is a proportional hazards model, which assumes

$$\alpha(t|\boldsymbol{x}) = \alpha_0(t) r(\boldsymbol{x}|\boldsymbol{\beta}), \tag{2.11}$$

{eq:PH}

where $\alpha_0(t)$ is an *unspecified* baseline hazard function shared between all individuals, and $r(\boldsymbol{x}|\beta)$ is a so-called relative risk function parameterized with regression coefficients $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)$. We choose $r(\boldsymbol{x})$ such that it is appropriately normalized, meaning $r(\mathbf{0}) = 1$. A crucial assumption here is that the effects of the covariates are fixed in time. In this setup, it turns out that we can do regression without specifying the baseline hazard. This is a major advantage, because we then do not have to think about modelling effects in time. Given data $D = (t_i, d_i, \boldsymbol{x}_i)_{i=1}^N$, we may set up a so-called partial likelihood (Cox, 1972).

The idea behind the partial likelihood is as follows. We have observed data D with at least some censoring indicators d_i equal to 1. In partial likelihood regression, we simply ignore the censored observations. Informally, the probability of the event happening at a time t for some individual j is the probability of an event happening to individual i at time t, divided by the total probability of an event happening at time t, the instantaneous probability of an event happening to that individual at that time, i.e., the hazard function of that individual at that time,

$$\frac{\Pr(\text{event happens to individual } i \text{ at time } t)}{\Pr(\text{event happens to } \textit{any} \text{ individual } j \text{ at time } t)}. \tag{2.12}$$

{eq:hazard-frac-informal}

More formally, we look at the instantaneous probability of an event happening for the individual i, which is the hazard function $\alpha(t|\boldsymbol{x}_i)$. Thus the total probability of an event happening at time t is the sum of the hazard functions of all individuals in the risk set $R(t)$, which, again, is defined as

$$R(t) = \{i \colon t_i \geq t \,, i = 1, 2, \ldots, n\}.$$

From all the uncensored observations, we know that events happened at times $\{t_i \colon d_i = 1, i = 1, 2, \ldots, N\}$. Therefore, we can construct expressions for (2.12) at all the observed, uncensored, times. Inserting the probabilities into the informal expression in (2.12), an individual with an observed event at t_i contribues to the likelihood with

$$\frac{\alpha_0(t_i) r(\boldsymbol{x}_i)}{\sum_{j \in R(t_i)} \alpha_0(t_i) r(\boldsymbol{x}_j)}. \tag{2.13}$$

{eq:hazard-frac}

Now, assuming that observations are independent and identically distributed, we say that the *partial likelihood* of the model given the observed data is the product of all ratios (2.13), namely

$$\mathrm{pl}(\boldsymbol{\beta}) = \prod_{i \colon d_i = 1} \frac{\alpha_0(t_i) r(\boldsymbol{x}_i)}{\sum_{j \in R(t_i)} \alpha_0(t_i) r(\boldsymbol{x}_j)} = \prod_{i \colon d_i = 1} \frac{\alpha_0(t_i) r(\boldsymbol{x}_i)}{\alpha_0(t_i) \sum_{j \in R(t_i)} r(\boldsymbol{x}_j)}. \tag{2.14}$$

{eq:pl}

In the expression above, (2.14), we rearrange the denominator such that the baseline hazard of individual i, $\alpha_0(t_i)$, is moved out of the sum. This hazard depends on individual i, while the sum is a sum over all individuals j. It therefore cancels out, and we are left with just the relative risk functions,

$$\mathrm{pl}(\boldsymbol{\beta}) = \prod_{i \colon d_i = 1} \frac{r(\boldsymbol{x}_i)}{\sum_{j \in R(t_i)} r(\boldsymbol{x}_j)}.$$

The fact that the baseline hazard cancels out is quite powerful. Even though proportional hazards regression is not fully parametric, the canceling means that we can model the effect of covariates in a fully parametric way, and without having to consider the baseline hazard.

Consider now possible choices for the relative risk function $r(\boldsymbol{x})$. The most common choice, by far, for $r(\boldsymbol{x})$ is

$$r(\boldsymbol{x}) = \exp(\boldsymbol{x}^T \boldsymbol{\beta}),$$

which leads to the famous Cox model (Cox, 1972). The Cox model is an attractive model because the value of the parameter $\boldsymbol{\beta}$ has a nice interpretation. Assume we have two observations with covariate vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, and that $\boldsymbol{x}_2$ is equal to $\boldsymbol{x}_1$ except for in element j, where $x_{2j} = x_{1j} + 1$. Then the ratio of the two hazard rates becomes

$$\frac{\exp(\boldsymbol{x}_2^T \boldsymbol{\beta})}{\exp(\boldsymbol{x}_1^T \boldsymbol{\beta})} = \exp((\boldsymbol{x}_2 - \boldsymbol{x}_1)^T \boldsymbol{\beta}) = \exp(\beta_j).$$

Furthermore, if the two covariate vectors differ in more elements, say, that, each element in $\boldsymbol{x}_2$ is one unit increased from each element in $\boldsymbol{x}_1$, we get that

$$\frac{\exp(\boldsymbol{x}_2^T \boldsymbol{\beta})}{\exp(\boldsymbol{x}_1^T \boldsymbol{\beta})} = \exp((\boldsymbol{x}_2 - \boldsymbol{x}_1)^T \boldsymbol{\beta}) = \exp(\beta_1 + \beta_2 + \ldots + \beta_p) = \exp(\beta_1) \exp(\beta_2) \cdots \exp(\beta_p).$$

In other words, each unit increase in a covariate is a multiplicative factor in the hazard.

subsec:cox-surv-prob

2.2.3 Estimating survival probabilities in a Cox model

For some applications, such as to evaluate the predictive power of a Cox model, we need the survival probability estimates of the model. In a Cox model, we do not estimate the baseline hazard (see subsection 2.2.2), but this is needed to get the actual survival probability estimates. In a Cox model which has estimated an additive predictor

$$\hat{\boldsymbol{\eta}} = \boldsymbol{x}\hat{\boldsymbol{\beta}}, \tag{2.15}$$

an estimate of the cumulative hazard $\hat{A}(t|\boldsymbol{x}_i)$ is given by

$$\hat{A}(t|\boldsymbol{x}_i) = \int_0^t \alpha_0(s)\exp(\hat{\boldsymbol{\eta}})\ \mathrm{d}s = \exp(\hat{\boldsymbol{\eta}})\hat{A}_0(t).$$

To provide a survival prediction, we must therefore also estimate the cumulative baseline hazard. It can be done by the Breslow-Aalen estimator (see e.g. Aalen et al. (2008)), given by

$$\hat{A}_0(t) = \int_0^t \frac{\sum_{i=1}^N \mathrm{d}N_i(s)}{\sum_{i=1}^N Y_i(s)\exp\left(\hat{\boldsymbol{\eta}}\right)}. \tag{2.16}$$

Here the notation $\mathrm{d}N_i(s)$ means the number of observed events in the infinitesimal interval $[s, s+\ \mathrm{d}s)$. It will be zero for most intervals, but 1 at an observed event. We do not have any tied event times in the dataset, i.e. events happening simultaneously. If we did, we would have to make some more considerations in the above, but it is not an issue. Further, here $Y_i(s)$ is an indicator function which is 1 if individual i is at risk at time s, and 0 if not. See B.2 in Appendix B for the R code we wrote to estimate this. Given the relationship (2.4), it follows that

$$S(t) = -\exp(A(t)),$$

and hence we get an estimate of the survival function of the Cox estimates by

$$\hat{S}(t) = -\exp(\hat{A}(t)) = -\exp\left(\hat{A}_0(t)\exp(\hat{\boldsymbol{\eta}})\right).$$

2.2.4 Issues with the proportional hazards assumption

When assuming (2.11), i.e. $\alpha(t|\boldsymbol{x}) = \alpha_0(t)r(x|\boldsymbol{\beta})$, we are making a relatively strong assumption. Namely, we assume that the ratio between the hazard function of two individuals is the same *at all times*, because the part of $\alpha(t|\boldsymbol{x})$ cancels out when we do regression, and because we assume that the covariate effects are constant in time, as $r(x|\boldsymbol{\beta})$ is not a function of time. This assumption goes under the name of the proportional hazards (PH) assumption. While, as we saw, this assumption greatly simplifies the inference, it is not necessarily satisfied in practice. In any case, it is very difficult to verify, in case of multivariate analysis, especially in high-dimensional contexts. In the literature, there exist alternative models, that do not require the PH assumption. One of these is Aalen's additive model, which is an example of additive hazard modelling. In Aalen's model, the hazard function takes the form

$$\alpha(t|\boldsymbol{x}) = \beta_0(t) + \beta_1(t)x_1(t) + \beta_2(t)x_2(t) + \ldots + \beta_p(t)x_p(t),$$

where $\beta_j(t), j = 1, \ldots, p$ is the increase in the hazard at time t corresponding to a unit's increase in the j-th covariate. Another alternative model to the Cox model is the first hitting threshold model. In this thesis we will focus on the latter, focusing on the analysis of high-dimensional data. To our best knowledge, this is the first study which tries to combine FHT models and high-dimensional data.

2.2.5 Cox, proportional hazards, and variable selection

The PH assumption (2.11) is very often not valid. Consider survival data with two covariates x_1 and x_2. With Cox regression, the hazard is

$$\alpha(t|x_{i,1}, x_{i,2}) = \alpha_0(t) \exp(\beta_1 x_{i,1} + \beta_2 x_{i,2}).$$

There is a model inconsistency problem here. If we only observe $x_{i,1}$, and calculate the hazard $\alpha(t|x_{i,1})$, then this will not be of Cox regression form, regardless of the distribution of $x_2|x_1$.

In addition, if there is perfect Cox structure given x_1 alone, and perfect Cox structure given x_2 alone, one almost never has a Cox model in the joint space (x_1, x_2). That is, one almost never has the proportional hazards assumption actually satisfied. However, in practice, Cox regression is robust and tends to work well.

Given that Cox sometimes leaves us wanting of a more flexible and more theoretically plausible model, there is room for other models. One may attempt to start the model building task at a deeper level, taking biologically plausible assumptions about the background processes associated with life lengths. This is where first-hitting-time models come in.

2.3 First hitting time models

sec:FHT

So far we have done regression by modelling the log-likelihood $l(\cdot)$ and the hazard rate $\alpha(\cdot)$. Therefore we have not thought much about how a time-to-event is *generated*, other than the fact that it arises from a survival distribution $S(\cdot)$ with a corresponding hazard function $\alpha(\cdot)$. In this context, an individual is alive at one time. Slightly later, it is either still alive, or it may have died.

Instead, another way to approach survival analysis is to model the process which generates the survival data. We can imagine that each individual has an underlying stochastic process $Y(t)$, which we call a health process. When this health process reaches a barrier, or an end state, the individual dies. This is a conceptually appealing framework: Instead of just being a stochastic lifetime with a binary status of an event having happened or not or alive, it introduces the concept of distance to the event. We may call the time the health process hits the barrier or the end state the *first hitting time* (FHT) of a boundary or threshold state, by sample paths of a stochastic health process. This health process may be observable, but for most purposes it will be latent, i.e., unobservable.

Many types of time-to-death data may, in fact, be interpreted as FHTs. FHT models have a long history of application in diverse fields, including medicine and engineering. They have been used to describe the length of a hospital stay (Whitmore, 1975; Eaton and Whitmore, 1977), to model the duration of a strike

(Lancaster, 1972), to estimate degradation in components (Whitmore, 1995), and to assess lung cancer risk in railroad workers (Lee et al., 2004).

Eaton and Whitmore (1977) discuss FHTs as a general model for hospital stay. Aalen and Gjessing (2001) provide an overview of much of this subject. Lawless (2011) provides a compact overview of the theory. Lee and Whitmore (2003) provides an overview of first hitting time models for survival and time-to-event data. There is a large literature dealing with theoretical and mathematical aspects of FHT models.

Models based on this view are called first hitting time models (FHT). FHT models were introduced in Whitmore (1986), see Lee and Whitmore (2006) for a complete overview. Note that these authors use the term threshold regression when referring to regression for first hitting time models. We have, following Caroni (2017), chosen to not use this term, as it is also used to refer to a well established, and quite different, topic in econometrics.

fht-idea

2.3.1 General idea

A first-hitting-time (FHT) model has two basic components:

1. A parent stochastic process $\{Y(t), t \in \mathcal{T}, y \in \mathcal{Y}\}$, with initial value $Y(0) = y_0$, where $\mathcal{T}$ is the time space and $\mathcal{Y}$ is the state space of the process.

2. A boundary set $\mathcal{B}$, where $\mathcal{B} \subset \mathcal{Y}$. This is at times also referred to as a barrier or a threshold, depending on which is most appropriate to the context.

The process $\{Y(t)\}$ may have a variety of properties, such as one or many dimensions, the Markov property, continuous or discrete paths, and monotonic sample paths. Whether the sample path of the parent process is observable or latent (unobservable) is an important distinguishing characteristic of the FHT model. Latent (unobservable) processes are the most common by far. The boundary set $\mathcal{B}$ may also have different features. The basic model assumes that $\mathcal{B}$ is fixed in time. Some applications may require dependencies on time, i.e., $\mathcal{B}(t)$, but this case will not be considered in this thesis, and so we write $\mathcal{B}$.

Taking the initial value $Y(0) = y_0$ of the process to lie outside the boundary set $\mathcal{B}$, the *first hitting time* of $\mathcal{B}$ is the random variable T defined as

$$T = \inf_t (t \colon Y(t) \in \mathcal{B}) \,. \tag{2.17}$$

{eq:fht-t}

Thus, the first hitting time is the time when the stochastic process first encounters the boundary set $\mathcal{B}$. The boundary set defines a stopping condition for the process. Note that when the parent process is latent, there is no direct way of observing the FHT even in the state space of the process.

In some versions of the FHT model, there is no guarantee that the process $\{Y(t)\}$ will reach the boundary set $\mathcal{B}$, so $P(T < \infty) < 1$. We will let $T = \infty$ denote the absence of a finite hitting time with

$$P(T = \infty) = 1 - P(T < \infty). \tag{2.18}$$

{eq:T-is-infinity}

This condition is sometimes plausible and a desirable model feature. Two common concepts in survival analysis might lead to this. One is the case of

competing risks. That is the case where an individual might die from not only one, but several causes, but we are only studying one of these. Naturally, then, we would not except that all individuals die from the cause that we study, and hence $P(T < \infty) < 1$. The second concept is related to "cure models." A cure model is a model where there are individuals with zero frailty, i.e., a nonsusceptible group (Aalen et al., 2008). Consult Maller and Zhou (1996) for more details.

2.3.2 Examples of first-hitting-time models

The parent stochastic process $Y(t)$ may take many forms, from Wiener processes to Markov chains. Similarly, the boundary state may also take various forms. For example, it may be a fixed threshold in a Wiener process or an absorbing state in a Markov chain. The freedom in the choice of these quantities show how flexible the FHT framework is. The choice of boundary must fit the process, such that the boundary set is a subset of the state space. We will now briefly mention some examples of choices of health processes and corresponding boundaries, before focusing on the most studied case, which uses as a choice of health process the Wiener process. These examples are taken from Lee and Whitmore (2006).

1. *Bernoulli process and negative binomial first hitting time.* The number of trials S required to reach the m-th success in a Bernoulli process $\{B_t, t = 1, 2, \ldots\}$ has a negative binomial distribution with parameters m and p, where p is the success probability of each trial. To give this setup our standard representation, we consider a parent process

$$\{Y(t),\, t = 0, 1, 2, \ldots\}$$

with initial value $Y_0 = y_0 = m$ and let

$$Y_t = y_0 - B_t,\, t = 1, 2, \ldots,$$

where $\{B_t\}$ is the aforementioned Bernoulli process. The first hitting time is the first Bernoulli trial $t = T$ for which Y_t equals 0. The number of rocket launches to get m satellites in orbit is an example of this FHT model.

2. *Gamma process and inverse gamma first hitting time.* Consider a parent process

$$\{Y(t),\, t \leq 0\}$$

with initial value $Y(0) = y_0 > 0$. Let

$$Y(t) = y_0 - Z(t),$$

where $Z(t)$, $t \leq 0$ is a gamma process with a scale parameter β, a shape parameter α and a starting point $Z(0) = 0$. The first hitting time of the zero level in the parent process ($Y = 0$) has an inverse gamma cumulative distribution function (cdf), defined by the identity

$$P(T > t) = P(Z(t) < y_0).$$

The identity follows from the fact that a gamma process has monotonic, nondecreasing sample paths. Computational routines for the gamma cdf allows the cdf of T to be computed readily. Singpurwalla (1995) and Lawless and Crowder (2004) consider the gamma process as a model for degradation. A gamma process is a good choice if we want a monotonic restriction on the movement of the health process (Lee and Whitmore, 2006). It might, for example, make sense if the health process is meant to model e.g. the breakdown of a structure.

3. *Poisson process and Erlang first hitting time.* The time S until the occurrence of the m-th event in a Poisson process

$$\{N(t),\, t \geq 0\}$$

with rate parameter λ has an Erlang distribution with parameters m and λ. Again, to give this setup our standard representation, we consider a parent process

$$\{Y(t),\, t \geq 0\}$$

with initial value $Y(0) = y_0 = m$ and let $Y(t) = y_0 - N(t$, where

$$\{N(t),\, t \geq 0\}$$

is the preceding Poisson process. The first hitting time is the earliest time $t = S$ when $Y(t) = 0$. This FHT model is illustrated by the time to failure of an engineering system consisting of m components in parallel, having identical and independent exponential lifetimes, that are placed in service successivelu as failures occur.

We are to choose a stochastic process to model a person's health process. A person's health, although generally decreasing over longer periods of time, will fluctuate, at times going up, and at times going down. If we consider the health process to be analogous with a person's health, it therefore makes sense to use a process which can both go up and down. Therefore the choice of a gamma process as the health process would not be appropriate.

The most usual setup for a first hitting time model is to have the health process be a Wiener process, and the boundary set be a fixed threshold level. This is attractive because it has closed-form probability and cumulative density functions, and its likelihood is computationally simple. There are also no restrictions on the movements of the process, meaning, it is non-monotonic. We will first give an introduction to the Wiener process.

subsec:wiener

2.3.3 Wiener process

Consider a continuous stochastic process $W(t)$ defined for $t \in [0, \infty)$, taking values in $\mathbb{R}$, and with initial value $W(0) = 0$. If W has increments that are independent and normally distributed with

$$\mathrm{E}[W(s+t) - W(t)] = 0 \text{ and } \mathrm{Var}[W(s+t) - W(t)] = s,$$

we call W a Wiener process. In other words, each increment has expectation 0 and has standard deviation proportional to the square root of the length of the

time interval. The position of the process at time t always follows a Gaussian distribution $N(0, \sqrt{t})$ (Aalen et al., 2008). To increase the flexibility of the Wiener process, we can introduce a new process Y,

$$Y(t) = y_0 + \mu t + \sigma W(t), \tag{2.19}$$ {wiener}

which is called a Wiener process with initial value y_0, drift coefficient μ, and diffusion coefficient σ. A good introduction to Wiener processes can be found in Cox and Miller (1965).

2.4 The first hitting time model with the Wiener process as health process

2.4.1 Wiener process as health process causes an inverse Gaussian first hitting time

We choose the stochastic process to be a Wiener process with intercept and drift (2.19), and we let the boundary be the non-positive numbers, $\mathcal{B} = (-\infty, 0]$. Then (2.17) becomes

$$T = \min_t \left(t \colon Y(t) < 0\right),$$

i.e. the first hitting time is the time it takes for the process to first reach a non-positive value. (Note that since the Wiener process is continuous, there is no difference between $\leq$ and $<$. We therefore use $<$ for convenience.) It can be shown that the first hitting time of such a Wiener process follows an inverse Gaussian distribution (Chhikara, 1988), with probability distribution function (pdf)

$$f(t|y_0, \mu, \sigma^2) = \frac{y_0}{\sqrt{2\pi\sigma^2 t^3}} \exp\left[-\frac{(\mu t + y_0)^2}{2\sigma^2 t}\right], \tag{2.20}$$ {eq:ig-pdf}

and cumulative distribution function (cdf)

$$F(t|\mu, \sigma^2, y_0) = \Phi\left[-\frac{\mu t + y_0}{\sqrt{\sigma^2 t}}\right] + \exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left[\frac{\mu t - y_0}{\sqrt{\sigma^2 t}}\right]. \tag{2.21}$$ {eq:ig-cdf}

As previously discussed in subsection 2.3.1, it is possible that the process does not reach the boundary set and thus cause an event. In this case, this means that the Wiener health process never reaches 0. This will happen if the drift μ is positive. If so, the probability distribution function in (2.20) is improper, and the probability of the time not being finite is

$$\Pr\left(T = \infty\right) = 1 - \Pr\left(T < \infty\right) = 1 - \exp\left(-2 \cdot y_0 \cdot \mu\right), \tag{2.22}$$ {eq:P-inf-FHT}

see Cox and Miller (1965).

Since we in survival analysis prefer working with the survival function $S(t) = 1 - F(t)$ rather than the cdf $F(t)$, we note that $S(t)$ becomes

$$S(t|\mu, \sigma^2, y_0) = \Phi\left[\frac{\mu t + y_0}{\sqrt{\sigma^2 t}}\right] - \exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left[\frac{\mu t - y_0}{\sqrt{\sigma^2 t}}\right], \tag{2.23}$$ {eq:ig-surv}

where $\Phi(x)$ is the cumulative distribution function of the standard normal,

$$\Phi(x) = \int_{-\infty}^{x} \phi(y)\ \mathrm{d}y,$$

and $\phi(x)$ is the pdf of the standard normal, defined as

$$\phi(x) = \frac{\exp\left(-x^2/2\right)}{\sqrt{2\pi}}.$$

Note that in (2.23) we used the fact that the cdf of the standard normal distribution is symmetric around 0, meaning that we were able to swap

$$1 - \Phi\left[-\frac{\mu t + y_0}{\sqrt{\sigma^2 t}}\right]$$

with

$$\Phi\left[\frac{\mu t + y_0}{\sqrt{\sigma^2 t}}\right].$$

Note that for the sake of brevity, when we talk about FHT from now on, we mean this particular FHT model, where the Wiener process is chosen as health process.

2.4.2 Effects of parameters in the Wiener process

Let us consider a a Wiener health process where its drift μ is strictly negative, and not zero. Such a health process will reach 0 with certainty, such that $P(T = \infty)$ (2.18) is 0. Clearly, for such a Wiener process, starting in $y_0 > 0$ and with a downwards drift $\mu < 0$, the movement is markedly in the direction of zero. If the variance σ^2 of the process is small in comparison to the drift, and the initial level is sufficiently large in comparison to σ^2, then the process $Y(t)$ will move in an almost straight line, such that

$$Y(t) \approx y_0 + \mu t.$$

Consequently, the first hitting time will be a near-deterministic function of y_0 and μ, such that

$$T \approx -\frac{y_0}{\mu}.$$

This is quite visible in Figure 2.4.2, which shows examples of 5 Wiener process paths with diffusion parameter $\sigma^2 = 1$, initial value $y_0 = 10$, and drift $\mu = -1$. Furthermore, if the diffusion is relatively small, such that $T \approx -y_0/\mu$, then either increasing y_0 or decreasing μ will have the same outcome, namely a larger lifetime T. Conversely, either choice of decreasing the initial level or increasing the drift will cause a smaller lifetime T. If σ^2 is relatively large compared to the drift, however, the diffusion part is more dominant and thus the hitting time is less predictable (Aalen et al., 2008).

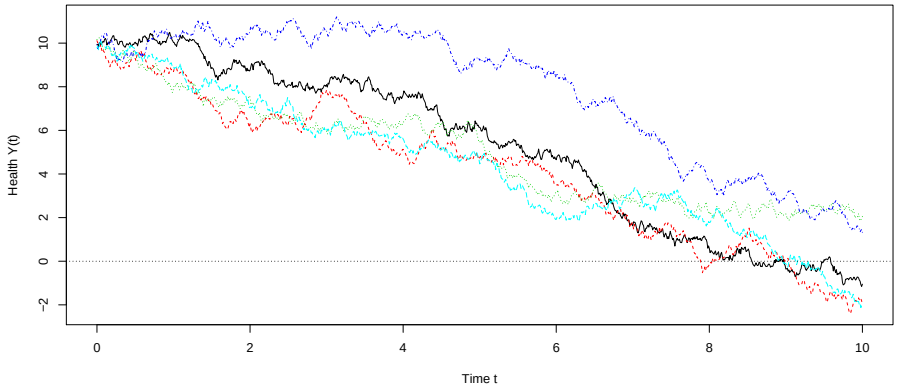
2.4.3 The inverse Gaussian parameterization with y_0, μ and σ is overdetermined

subsec: overdetermined

There are three parameters in the inverse Gaussian distribution, namely y_0, μ and σ. We observe, however, that both the pdf $f(t|y_0, \mu, \sigma^2)$ in (2.20) and the survival function $S(t|\mu, \sigma^2, y_0)$ in (2.23) only depend on these parameters through the ratios μ/σ and y_0/σ. Hence, in reality there are only two free parameters. In other words, we can without loss of generality fix one parameter. The conventional way to proceed is to set σ equal to 1 (Lee and Whitmore, 2006), and in this thesis we follow this convention.

plot:wiener

Figure 2.1: Example of 5 Wiener process paths with initial value $y_0 = 10$ and drift $\mu = -1$. The dashed horizontal line is at 0, at which point the process would cause an event in an FHT model.



2.4.4 The shape of the hazard function in the inverse Gaussian FHT model

From (2.1), we know that the hazard function can be seen as

$$\alpha(t) = f(t)/S(t).$$

If we plug in the inverse Gaussian versions of $f(t)$ and $S(t)$, we would get a rather intractable expression. However, we can make some comments on its shape as it relates to different values of y_0 and μ. Consider the shape of the hazard rate in three different cases for the initial level y_0:

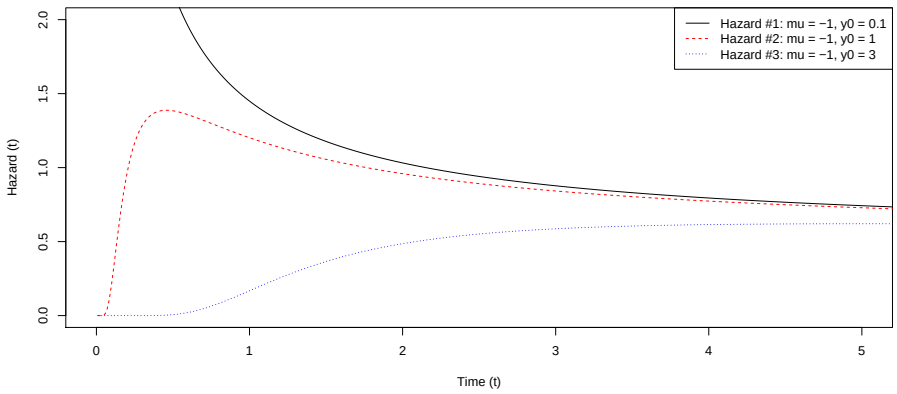
1. If y_0 is close to zero, we essentially get a decreasing hazard rate.
2. If y_0 is far from zero, however, we essentially get an increasing hazard rate.
3. If y_0 is somewhat inbetween, we get a hazard rate which first increases and then decreases (Aalen et al., 2008).

These three examples are clearly observed in Figure 2.4.4, where we have plotted the hazard function arising from three FHT models. All three have drift $\mu = -1$, but they have different initial levels y_0. The upper curve has $y_0 = 0.1$, and it is essentially decreasing, so it corresponds to the first point in the list above. The bottom curve has $y_0 = 3$, and it is essentially increasing, i.e. it corresponds to the second point in the list. The middle curve has $y_0 = 1$, and it first increases to a pronounced peak, before it decreases. Hence it corresponds to the last point in the list above.

Figure 2.4.4 shows an additional three hazard curves. These are attained by changing their drifts μ, but they have the same initial levels y_0, namely $y_0 = 1$. The upper curve has a larger drift, $\mu = -1$, while the middle has a drift of -0.5, and the lower curve has a drift of -0.25. We observe that as we increase

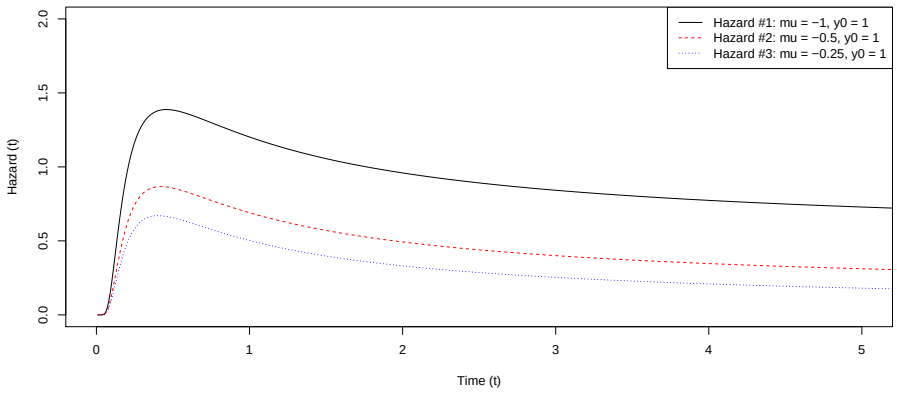
plot:hazards1

Figure 2.2: Examples of hazard functions arising from FHT models with Wiener processes. These processes have the same drift μ, but different initial levels y_0.



plot:hazards2

Figure 2.3: Examples of hazard functions arising from FHT models with Wiener processes. These processes have different drift parameters μ, but the same initial level y_0.



the drift in absolute terms, i.e. letting the process move quicker towards 0, the hazard is increased. These curves have the same shape, with a peak, but this peak is more pronounced the larger the drift. We conclude from these two example plots that y_0 has an impact on the shape, whereas μ mostly affects the height of the peak, i.e. the intensity of the hazard.

We note from this investigation that we do not have proportional hazards, as we would have been limited to in the case of a Cox model. However, the hazards do converge towards a value, in which case the hazard curves will be proportional. We see that the FHT framework with a Wiener process is a highly flexible parametric model for survival analysis. Indeed, much more flexible than Cox regression, since the hazard ratios in Cox are all confined to be constant over time.

2.5 Regression with Wiener first-hitting-time models

subsec:IG-reg

2.5.1 Structure of the additive predictors

We may introduce effects from covariates by allowing μ and y_0 to depend on covariates $\boldsymbol{x}$ and $\boldsymbol{z}$. A simple and much used setup (Lee and Whitmore, 2006; Caroni, 2017) is to use the identity link function for the drift μ,

$$\mu(\boldsymbol{\beta}) = \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x} = \beta_0 + \sum_{j=1}^{p_1} \beta_j x_j, \tag{2.24}$$

{eq:y0}

and the logarithm link function for the initial level y_0, since y_0 must be positive in our framework,

$$y_0(\boldsymbol{\gamma}) = \exp(\boldsymbol{\gamma}^{\mathrm{T}}\boldsymbol{z}) \Rightarrow \ln y_0(\boldsymbol{\gamma}) = \boldsymbol{\gamma}^{\mathrm{T}}\boldsymbol{z} = \gamma_0 + \sum_{j=1}^{p_2} \gamma_j z_j. \tag{2.25}$$

{eq:mu}

Here $\boldsymbol{\beta} \in \mathbb{R}^{p_1+1}$ and $\boldsymbol{\gamma} \in \mathbb{R}^{p_2+1}$ are vectors of regression coefficients. Note that we use separate names for the vectors $\boldsymbol{x}$ and $\boldsymbol{z}$ corresponding to μ and y_0, respectively. We may let these share none, some, or all elements. For the applications discussed later in this thesis, we let these covariate vectors be entirely separate.

Plugging in the pdf (2.20) and the survival function (2.23) into the log-likelihood (2.9), we get that the log-likelihood of a survival data set with the inverse Gaussian FHT model, is

$$\begin{aligned} l(y_0, \mu, \sigma) = \sum_{i=1}^{n} d_i \Bigg(& \ln y_0 - \frac{1}{2}\ln\!\left(2\pi\sigma^2 t_i^3\right) - \frac{(\mu t_i + y_0)^2}{2\sigma^2 t_i} \Bigg) \\ & + (1-d_i)\ln\!\left(\Phi\!\left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}}\right) - \exp\!\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\!\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right)\right). \end{aligned} \tag{2.26}$$

{eq:loglik}

2.5.2 State of the art

At the moment, the standard for fitting an inverse Gaussian FHT model to survival data is to use numerical likelihood maximization (Caroni, 2017). A few

software packages for performing this maximization exist. For `R` (R Core Team, 2013), the `threg` package exist (Xiao et al., 2015). However, the software "only" performs unpenalized maximization likelihood estimation. In modern data sets, it is necessary to perform some sort of penalized or regularized estimation. It is also necessary to be able to perform variable selection, as in the case of using data sets containing genetic information, the number of covariates will typically be far larger than the number of individual observations. To the best of our knowledge, there does not exist any such method for FHT models.

2.5.3 Constructing survival probabilities based on estimates

Consider a case where we have performed estimation of an FHT model, such that we have estimated parameters $\hat{y}_0$ and $\hat{\mu}$. Since we have a parametric expression for the survival function $S(t)$ (2.23), we can construct estimates of the survival probability at time t, by plugging in these estimates. Recall that σ^2 is 1, so it is removed from the expression. We denote the estimated probability for $\hat{S}(t)$,

$$\hat{S}(t|\hat{\mu}, \hat{y}_0) = \Phi[\hat{\mu}t + \hat{y}_0] - \exp(-2 \cdot \hat{y}_0 \cdot \hat{\mu})\Phi[\hat{\mu}t - \hat{y}_0].$$

With an estimate of the probability of an individual at time t, we are for example able to use the Brier score to assess the predictive performance of the estimated model. We will discuss the Brier score later in the thesis, in section 5.3.

2.5.4 Combining clinical and genetic data in the inverse Gaussian FHT model

subsec:FHT-combine

Classical survival analysis models have considered a small number of predictors, and they have often been clinical variables. In recent years, much effort has been put into being able to use genetic information to make predictions of survival, and it has become cheap and feasible to obtain genetic data. Rather than only using genetic data in models, it would be best to be able to combine such data with clinical data and other types of data, which might typically be shown to be predictive. There exist various schemes for making such models in Cox regression settings, see e.g. Frigessi et al. (2007). It is, however, not straightforward to perform such a combination. A naive way to combine these is to simply merge the clinical data and genetic data into one group. We subsequently standardize the data, as we often need to do for proper model performance, but then we might very well miss important effects of the genomic data. Some more sophisticated approaches used require tuning and weighting of parameters (De Bin et al., 2014).

The FHT model, however, lends itself nicely to combining clinical and genetic data. Aalen and Gjessing (2001) suggest to make an a priori splitting of the covariates into two groups. One group is related to "fixed" covariates, e.g. genes, and the other group of covariates is related to "lifestyle," e.g., indicators about smoking, or measurements, such as weight. If we incorporate this split of covariates in FHT models, it seems reasonable to let the initial level y_0 of the health process be a function of the "fixed" covariates, while letting the drift μ be a function of the "lifestyle" covariates. With the fact that the two groups relate to two separate parameters, they can be standardized separately, and thus we might more easily be able to extract explanatory power from both the genetic data and the clinical data.

Chapter 3

Gradient boosting

ch:boosting

Boosting is one of the most promising methodological approaches for data analysis developed in the last two decades (Mayr et al., 2014a). It has become a staple part of the statistical learning toolbox because it is a flexible tool for estimating interpretable statistical models. Boosting, however, originated as a black box algorithm in the fields of computational learning theory and machine learning, not in statistics.

Computer scientists Michael Kearns and Leslie Valiant, who were working on computational learning theory, posed the following question (Kearns and Valiant, 1989): Could any weak learner be transformed to become a strong learner? A weak learner, sometimes also simple or base learner, is a learner that has a low performance. For example, in the context of classification, a weak learner is one that performs only slightly better than random (uniform) chance. In the binary classification setting, it would only perform slightly better than a coin flip. Meanwhile, a strong learner should be able to perform in a near-perfect fashion, for example attaining high accuracy on a prediction task. We will first give a summary of the history of boosting, starting with AdaBoost (Freund and Schapire, 1996), an algorithm that proved that the answer to the original question above was yes. For a complete overview of the history of boosting, see Mayr et al. (2014a,b, 2017).

3.1 AdaBoost: From machine learning to statistical boosting

The original AdaBoost, also called Discrete AdaBoost (Freund and Schapire, 1996) is an iterative algorithm for constructing a binary classifier $F(\cdot)$. It was the first *adaptive* boosting algorithm, as it automatically adjusted its parameters to the data based on its perfomance. In the binary classification problem, we are given a set of observations

$$D = (\boldsymbol{x}_i, y_i)_{i=1}^N,$$

where $\boldsymbol{x}_i \in \mathbb{R}^p$ is a vector of covariates, and $y_i \in \{-1, 1\}$ is a binary response, i.e., positive or negative; yes or no. We want to find a rule which best separates these observations into the correct classes $\{-1, 1\}$, as well as being able to classify new, unseen observations of the same form. Some observations are hard to classify, whereas some are not. Freund and Schapire (1996) proposed that

one could estimate several classifiers, and assign a weight α_m to each classifier. By giving more weight to a more accurate classifier, a weighted sum of all of these classifications might be a good classification. It turns out that this is the case. We now give an explanation of the algorithm.

AdaBoost is an iterative algorithm. In a given step m, we use a weak learner $h(\cdot)$ to estimate a classifier $\hat{h}^{[m]}(\cdot)$, that minimizes the weighted sum of misclassified points. Based on the misclassification rate a weight $\alpha^{[m]}$ is assigned to the classifier $\hat{h}^{[m]}(\cdot)$. After a first iteration, the classifier is $F^{[1]}(\cdot) = \hat{\alpha}^{[1]}\hat{h}^{[1]}(\cdot)$. Using this initial classifier, some points will be correctly classified, and some will be misclassified. We increase the weights of the misclassified ones, and normalize the weights afterwards, to ensure that the sum of the weights is always the same. This results in the correctly classified observations having a reduced weight, and with misclassified observations having an increased weight. In the following iteration, we apply again a weak learner which minimizes the weighted sum of the observations, and we reweight the observations accordingly, in the same manner as before. Again, calculate a weight to give to this new classifier, and add it to the previous classifier, such that $F^{[2]}(\cdot) = \hat{\alpha}^{[1]}\hat{h}^{[1]}(\cdot) + \alpha^{[2]}h^{[2]}(\cdot)$. Continue iterating in this fashion until an iteration number m_{stop} is reached. The resulting AdaBoost classifier $\hat{F}(\cdot)$ becomes

$$\hat{F}(\cdot) = F^{[m_{\text{stop}}]}(\cdot) = \sum_{m=1}^{m_{\text{stop}}} \hat{\alpha}^{[m]}\hat{h}^{[m]}(\cdot),$$

i.e. a linear combination of the weak classifiers, or in essence a weighted majority vote of weak learners given the observations.

The AdaBoost algorithm often carries out highly accurate prediction. In practice, it is often used with stumps, which are decision trees with one split. For example, Bauer and Kohavi (1999) report an average 27% relative improvement in the misclassification error for AdaBoost using stump trees, compared to the error attained with a single decision tree. They conclude that boosting not only reduces the variance in the prediction error from using different training data sets, but that it is also able to reduce the average difference between the predicted and the true class, i.e., the bias. Breiman (1998) supports this analysis. Because of its plug-and-play nature and the fact that it never seemed to overfit, which occurs when the learned classifier degrades in test error because of being too specialized on its training set, Breiman remarked that "boosting is the best off-the-shelf classifier in the world" (Hastie et al., 2009).

While originally developed for binary classification, boosting is now used to estimate the unknown quantities in more general statistical models and settings. In the following we present boosting in a more general statistical regression scheme. Moreover, In its original formulation the AdaBoost classifier does not have interpretable coefficients, and as such it is a so-called black-box algorithm. This means that we are unable to infer anything about the effect of different covariates. In statistics, however, we are interested in models which are interpretable. In the rest of this chapter, we will discuss gradient boosting algorithms. We will start by defining the problem such algorithms try to solve, and introduce some notation. We will then explain the gradient descent algorithm, and, more precisely, gradient boosting.

3.2 General model structure, setting, and chosen notation

The aim of statistical boosting algorithms is to estimate and select the effects in structured additive regression models. Consider a data set

$$D = (\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=1}^N$$

containing the values of an outcome variable $\boldsymbol{y}$ and predictor variables $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_p$, forming covariate matrix $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_p)$. We assume that the samples $i = 1, \ldots, N$ are independently generated from an identical distribution over the joint space $\mathcal{X} \times \mathcal{Y}$. The input space of $\boldsymbol{x}$ is a possibly high-dimensional $\mathcal{X} \in \mathbb{R}^p$ and the output space is a low-dimensional space $\mathcal{Y}$. For the majority of applications, the output space $\mathcal{Y}$ is one-dimensional, but we will explicitly allow for multidimensional outcome variables. Our objective is to model the relationship between $\boldsymbol{y}$ and $\boldsymbol{x}$ and to obtain an "optimal" prediction of $\boldsymbol{y}$ given $\boldsymbol{x}$. To model the relationship, we will use an approach which very similar to the generalized additive model (GAM) approach (Hastie and Tibshirani, 1990). We will assume that the conditional outcome $\boldsymbol{y}|\boldsymbol{x}$ follows some probability distribution function (pdf)

$$\psi(\boldsymbol{y}|\theta(\boldsymbol{x})), \tag{3.1}$$ {eq:psi}

where θ is a parameter in the distribution function, typically related to the mean. We will at times refer to ψ as a prediction function, when we use it to estimate parameters. Further, we will model the distribution parameter θ as a functional of the covariates $\boldsymbol{X}$, with conditional expectation given the observed value $\boldsymbol{x}$ as

$$g(\mathbb{E}(\theta(\boldsymbol{x})) = f(\boldsymbol{x}), \tag{3.2}$$

where $g(\cdot)$ is a so-called link function and $f(\cdot)$ is a predictor. We will discuss $f(\cdot)$ shortly. We observe that if we use $g^{-1}(\cdot)$, i.e. the inverse of the link function, on this expression, we get

$$\mathbb{E}(\theta(\boldsymbol{x})) = g^{-1}(f(\boldsymbol{x})).$$

This means that the conditional expectation of θ given the observed $\boldsymbol{x}$ is a transformation of the additive predictor $f(\boldsymbol{x})$ using the inverse of the link function. The link function will be chosen appropriately for the parameter θ in the distribution ψ, and is typically used to constrain the domain of the parameter. For example, if we choose the logarithm as the link function, the inverse link function is the exponential function, meaning that

$$\mathbb{E}(\theta(\boldsymbol{x})) = \exp(f(\boldsymbol{x})), \tag{3.3}$$

which will constrain the expectation to be a positive number.

The predictor $f(\cdot)$ can be modeled in many ways. A common model is to let it be an *additive* predictor, consisting of additive effects of single predictors. This is called a generalized additive model (GAM), and it is specified by

$$f(\boldsymbol{x}) = \beta_0 + f_1(x_1) + \ldots + f_p(x_p), \tag{3.4}$$ {eq:gam}

where β_0 is a common intercept and the functions $f_j(x_j), j = 1, \ldots, p$ are single predictors, which are partial effects of the variables x_j. The generic notation

$$f_j(x_j)$$

may represent different types of predictor effects, such as classical linear effects $x_j\beta_j$, smooth non-linear effects constructed via regression splines, spatial effects or random effects of the explanatory variable x_j, and so on. The component-wise effects will typically be built up by additive estimation of base-learners, and statistical boosting is one way to perform this additive estimation. Statistical boosting algorithms are one way to estimate such models. These algorithms typically estimate $f(\boldsymbol{x})$ by estimating component-wise effects for each component j, and these are in turn build up by estimation of base-learners $h_1(\cdot), \ldots, h_p(\cdot)$. We will discuss this more in Section 3.5, which introduces component-wise boosting.

See comments from Riccardo: Explain this better!

We evaluate the fit of a model ϕ and its additive predictor $f(\cdot)$ by using a loss function $\rho(y, f(\cdot))$. The loss function is a measure of the discrepancy between the observed outcome $\boldsymbol{y}$ and the additive predictor $f(\cdot)$. In machine learning and optimization, one usually talks of loss functions, and as the name suggests, we wish to minimize the loss. We will use the negative log-likelihood of the distribution of the response as a loss function because it is common in statistics (Mayr et al., 2014a). In these cases, the loss function, which works on one set of observations $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, is

$$\rho(\boldsymbol{y}_i, \theta(\boldsymbol{x}_i)) = -\log \psi(\boldsymbol{y}_i | \theta(\boldsymbol{x}_i)),$$

since the likelihood of one observation is simply the distribution given the observed data. Note that there is a theoretical result stating that maximizing the log-likelihood is equivalent to minimizing the Kullback-Leibler Divergence, which is a measure of the difference between the distribution of the data itself and the assumed distribution ψ (Akaike, 1998). This shows that, having assumed a distribution ψ for the responses, it is a good choice of loss function to use the log-likelihood of ψ.

3.2.1 Example of a model and corresponding loss function

subsec:model-example

Let us consider at a specific example of a distribution and a loss function. We have a dataset $D = (\boldsymbol{x}_i, y_i)_{i=1}^N$ where the responses y_i are continuous and univariate. We further assume that the responses follow a normal distribution, conditioned on the data. Thus we wish to model the conditional mean $\mu(\boldsymbol{x})$, and so we use μ instead of θ. Since the responses are continuous and normal, we do not need any transformation of the additive predictor, which means that the link function is the identity function,

$$g(\boldsymbol{x}) = \boldsymbol{x}.$$

Further, it means that

$$\mathbb{E}(\mu(\boldsymbol{x})) = f(\boldsymbol{x}).$$

For a normally distributed observation y, the likelihood is the familiar pdf,

$$f(y|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{(y-\mu)^2}{2\sigma^2} \right\},$$

and we derive the loss function ρ accordingly, yielding

$$\begin{aligned}\rho(y, \mu(\boldsymbol{x})) &= -\log f(y|\mu(\boldsymbol{x})) \\ &= \log(\sqrt{2\pi\sigma^2}) + \frac{(y-\mu(\boldsymbol{x}))^2}{2\sigma^2} \\ &\propto (y-\mu(\boldsymbol{x}))^2,\end{aligned}$$

which is the familiar L_2 loss function. Note that since we will only model $\mu(\cdot)$, the loss function need not depend on σ^2. Similarly we have disregarded all proportionality constants. With all those parts in place, we can model the additive predictor $f(\cdot)$. One way to do this is by gradient boosting, which we will discuss soon.

3.2.2 Model selection and model assessment

Having chosen a distribution ψ and a loss function ρ, we wish to find the parameter θ which minimizes the loss function of all unseen data

$$(\boldsymbol{X}, Y)\,.$$

This means that we wish to minimize

$$\mathrm{Err}(\theta) = \mathbb{E}_{Y,X}\left[\rho(Y, \theta(\boldsymbol{X}))\right]. \tag{3.5}$$

{eq:unseen-error}

However, we are unable to estimate this quantity, as we do not have access to all such unseen data. We therefore need a so-called *test set*

$$D_{\text{test}} = (\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=1}^{N_{\text{test}}}.$$

We can then calculate a test error Err of a specific θ, which is the mean of the loss using θ, over all observations in the test set D_{test},

$$\mathrm{Err}_{D_{\text{test}}}(\theta) = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \rho(\boldsymbol{y}_i, \theta(\boldsymbol{x}_i)). \tag{3.6}$$

{eq:test-error}

$\mathrm{Err}_{D_{\text{test}}}$ is an estimate of Err, since D_{test} is a realization of unseen data. To ensure that $\mathrm{Err}_{D_{\text{test}}}$ is an unbiased estimate, we will not use any data from the test set to estimate θ. To estimate θ, we will therefore use a so-called *training set*, which we will denote

$$D_{\text{train}} = (\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=1}^{N_{\text{train}}}.$$

We can calculate the training error $\overline{\mathrm{err}}(\theta)$, also called the *in-sample error*, or the *empirical risk*, which similarly is the sum of the loss function over all observations in the training set,

$$\overline{\mathrm{err}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \rho(\boldsymbol{y}_i, \theta(\boldsymbol{x}_i)).$$

Our goal will be to use gradient boosting to minimize 3.6. To understand gradient boosting, we first need to understand the gradient descent algorithm.

3.3 Gradient descent

Suppose we are trying to minimize a differentiable multivariate function $G\colon \mathbb{R}^p \to \mathbb{R}$, where $p \in \mathbb{N}$. Gradient descent is a so-called greedy algorithm for finding the minimum of such a function G, and one which is quite simple and surprisingly effective. If all partial derivatives of G at a point

$$\boldsymbol{x}^{[0]} = \left(x_1^{[0]}, x_2^{[0]}, \ldots, x_n^{[0]}\right) \in \mathbb{R}^p$$

exist, then the gradient of G at $\boldsymbol{x}^{[0]}$ is the vector of all its partial derivatives at $\boldsymbol{x}^{[0]}$, namely

$$\nabla G\left(\boldsymbol{x}^{[0]}\right) = \left(\frac{\partial G(\boldsymbol{x}^{[0]})}{\partial x_1}, \frac{\partial G(\boldsymbol{x}^{[0]})}{\partial x_2}, \ldots, \frac{\partial G(\boldsymbol{x}^{[0]})}{\partial x_p}\right).$$

The motivation behind the gradient descent algorithm is that in a small interval around the point $\boldsymbol{x}_0$, G is most decreasing in the direction of the negative gradient at that point. Therefore, if we take a small step slightly in the direction of the negative gradient at $\boldsymbol{x}^{[0]}$, which we denote $\boldsymbol{g}^{[0]}$, from $\boldsymbol{x}^{[0]}$ to a new value $\boldsymbol{x}^{[1]}$, we end up with a slightly lower function value: The new function value $G(\boldsymbol{x}^{[1]})$ will be smaller than $G(\boldsymbol{x}^{[0]})$. The algorithm is greedy because it always goes in a direction which is immediately better. The length $a^{[1]}$ of this small step is found by a line search, i.e., by finding the step length which gives the best $G(\boldsymbol{x}^{[1]})$, where then the new point is

$$\boldsymbol{x}^{[1]} = \boldsymbol{x}^{[0]} + a^{[1]} \cdot \boldsymbol{g}^{[0]}.$$

By repeating this procedure m_{stop} number of times, the point $\boldsymbol{x}^{[m_{\text{stop}}]}$ will yield a minimum value of $G(\cdot)$. The number of iterations m_{stop} is decided by stopping when the decrease in function value is smaller than some pre-specified threshold $\epsilon > 0$,

$$m_{\text{stop}} = \min_m G(\boldsymbol{x}^{[m-1]}) - G(\boldsymbol{x}^{[m]}) < \epsilon.$$

With sufficiently small steps, gradient descent will always converge, albeit possibly to a local minimum. For a schematic overview of the algorithm, see Algorithm 1.

The gradient descent algorithm is surprisingly robust. Even though it may converge to a local minimum, it seems to often find good solutions globally. This is likely related to research which has found that in high-dimensional spaces, most minima are not minima, but in fact, saddlepoints masquerading as local minima (Dauphin et al., 2014). This means that the size of the improvements in the function value $G(\cdot)$ will decrease since the gradient will be small at this saddlepoint. When using a gradient descent method one typically sets a threshold ϵ at which the algorithm terminates when the gradient becomes smaller than the threshold, as we did. However if the algorithm continues for a long enough time, then the multivariate gradient descent search should be able to continue to decrease the function value after it “escapes” the saddle point.

algo:grad-desc

Algorithm 1 Gradient descent

We wish to minimize $G(\boldsymbol{x})$, i.e. solve $\min_{\boldsymbol{x}} G(\boldsymbol{x})$, where G is a multivariate function $G\colon \mathbb{R}^p \to \mathbb{R}$.

1. Start with an initial guess $\boldsymbol{x}^{[0]} \in \mathbb{R}^p$, for example $\boldsymbol{x}^{[0]} = \mathbf{0}$, and set the number of iterations m to 0.

grad-desc-iter

2. Increase the iteration number m by 1.

3. Calculate the direction to step in, i.e., the derivative at the current point,
$$\boldsymbol{g}^{[m-1]} = -\nabla G(\boldsymbol{x}^{[m-1]}).$$

4. Solve the line search to find the best step length $a^{[m]}$,
$$a^{[m]} = \operatorname*{argmin}_a \boldsymbol{x}^{[m-1]} + a \cdot \boldsymbol{g}^{[m-1]}.$$

5. The step in iteration m becomes
$$\boldsymbol{h}_m = a^{[m]} \cdot \boldsymbol{g}^{[m-1]}.$$

6. Let $\boldsymbol{x}^{[m]} = \boldsymbol{x}^{[m-1]} + \boldsymbol{h}^{[m-1]}$.

7. Decide if the algorithm should terminate by checking if the decrease in function value is smaller than the pre-specified threshold,
$$G(\boldsymbol{x}^{[m-1]}) - G(\boldsymbol{x}^{[m]}) < \epsilon.$$
If this is true, set the number of iterations m_{stop} to m, and go to the next step of the algorithm. If not, go to step 2.

8. The resulting minimum point for $G(\cdot)$ is
$$\boldsymbol{x}^{[m_{\text{stop}}]} = \boldsymbol{x}^{[0]} + \sum_{m=1}^{m_{\text{stop}}} \boldsymbol{h}^{[m]}.$$

3.4 The gradient boosting approach

3.4.1 Direct gradient descent on the loss function

In a seminal paper, Friedman (2001) developed an iterative algorithm for fitting a predictor $f(\boldsymbol{x})$, and he called the algorithm gradient boosting. He showed that AdaBoost performs this algorithm for a particular loss function, namely the exponential loss function. See Hastie et al. (2009) for a good demonstration of Friedman's argument. With his work, Friedman provided a way of viewing boosting through a statistical lens, and connected the successful machine learning approach to the world of statistical modelling.

The key idea of gradient boosting is to iteratively fit the different predictors with simple functions (base-learners) and combine the estimates into a predictor. The base-learners are in particular fitted to the negative gradient of the loss function.

Consider data as in the example in subsection 3.2.1. We have covariate vectors $\boldsymbol{x}_i$ and continuous responses y_i, where $i = 1, 2, \ldots, N$. We assume that the conditional response y_i, given $\boldsymbol{x}_i$, follows a distribution $\psi(\theta)$ with a parameter θ, which we will let depend on $\boldsymbol{x}_i$, i.e., $\theta(\boldsymbol{x}_i)$. Based on the distribution ψ, we derive a loss function ρ, as seen in Subsection 3.2.1. We wish to estimate the $\hat{\theta}(\cdot)$ that minimizes the training error, the empirical risk over the observed training data set

$$\underset{\hat{\theta}}{\operatorname{argmin}}\, \overline{\operatorname{err}}(\theta) = \underset{\hat{\theta}}{\operatorname{argmin}}\, \frac{1}{N} \sum_{i=1}^{N} \rho(\boldsymbol{y}_i, \theta(\boldsymbol{x}_i)).$$

We can think of the empirical risk $\overline{\operatorname{err}}(\theta)$ as a multivariate function $\overline{\operatorname{err}}(\boldsymbol{\theta})$, where the variables of the function are the parameter values of θ at each point $\boldsymbol{x}_i$,

$$\boldsymbol{\theta}(\boldsymbol{x}) = (\theta(\boldsymbol{x}_1),\, \theta(\boldsymbol{x}_2),\, \ldots,\, \theta(\boldsymbol{x}_N))\,.$$

These are plugged into the loss function for their corresponding observations, i.e.,

$$\overline{\operatorname{err}}(\boldsymbol{\theta}(\boldsymbol{x})) = \overline{\operatorname{err}}(\theta(\boldsymbol{x}_1),\, \theta(\boldsymbol{x}_2),\, \ldots,\, \theta(\boldsymbol{x}_N)) = \frac{1}{N} \sum_{i=1}^{N} \rho(\boldsymbol{y}_i, \theta(\boldsymbol{x}_i)).$$

In this view, $\overline{\operatorname{err}}$ is a multivariate function $\overline{\operatorname{err}}\colon \mathbb{R}^N \to \mathbb{R}$. We can therefore use gradient descent (see the previous section) directly on this function. To do so, we consider $\hat{\theta}(\boldsymbol{x}_i)$ as a sum

$$\hat{\theta}(\boldsymbol{x}_i) = \beta_0 + \sum_{m=1}^{m_{\text{stop}}} f^{[m]}(\boldsymbol{x}_i),$$

where the first term, β_0, is an initial guess which is common for all $\boldsymbol{x}_i$, and the remaining $\{\hat{f}^{[m]}(\boldsymbol{x}_i)\}_{m=1}^{M}$ function values are increments – steps, or boosts. The initial guess β_0 should be the constant which minimizes the loss function ρ, e.g. the maxmimum likelihood constant in cases where the loss function is a negative log-likelihood.

To perform gradient descent on the $\overline{\text{err}}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N)$, we need to calculate its negative gradient. The negative partial derivative for each observation is

$$-\frac{\partial}{\partial\theta(\boldsymbol{x}_i)}\rho(y_i, \theta(\boldsymbol{x}_i)).$$

Thus the gradient $\nabla\overline{\text{err}}(\boldsymbol{\theta})$, which we will denote $\boldsymbol{u}$, and call generalized residuals, becomes

$$\boldsymbol{u} = (u_i)_{i=1}^N = \left(-\frac{\partial}{\partial\theta(\boldsymbol{x}_i)}\rho(y_i, \hat{\theta}(\boldsymbol{x}_i))\right)_{i=1}^N.$$

Since we now know how to calculate the gradient of a vector of estimated function values, we can constuct an algorithm for performing gradient descent on the loss function. We initialize $\hat{\theta}^{[0]}(\boldsymbol{x}_i)$ to β_0 for all $\boldsymbol{x}_i$, where β_0 is found by

$$\beta_0 = \underset{c}{\text{argmin}}\, \overline{\text{err}}(c).$$

We then iterate. In a step $m > 0$, we calculate the negative gradient $\boldsymbol{u}^{[m-1]}$, and perform a gradient descent step in the direction $\boldsymbol{u}^{[m-1]}$. The gradient descent step is

$$\hat{\theta}^{[m]}(\boldsymbol{x}_i) = \hat{\theta}^{[m-1]}(\boldsymbol{x}_i) + a^{[m]} \cdot u_i^{[m-1]},$$

for each observation $\boldsymbol{x}_i$. Thus in vector form, the step is

$$\hat{\boldsymbol{\theta}}^{[m]}(\boldsymbol{x}) = \hat{\boldsymbol{\theta}}^{[m-1]}(\boldsymbol{x}) + a^{[m]} \cdot \boldsymbol{u}^{[m-1]},$$

where the step length $a^{[m]}$ is found by a line search, as before. After m_{stop} number of steps, we will have converged to a solution $\hat{\boldsymbol{\theta}}^{[m_{\text{stop}}]}$,

$$\hat{\boldsymbol{\theta}}^{[m_{\text{stop}}]} = \hat{\theta}^{[m_{\text{stop}}]}(\boldsymbol{x}_1), \hat{\theta}^{[m_{\text{stop}}]}(\boldsymbol{x}_2), \ldots, \hat{\theta}^{[m_{\text{stop}}]}(\boldsymbol{x}_N),$$

which is a minimizer for the loss function $\overline{\text{err}}$. This nonparametric approach would reduce the error of each data point. However, it will not generalize to a similar data set where one observes different data points, since we only considers the points $\boldsymbol{x}_i$ in the training set. We therefore need to do something else, and this is the functional gradient descent algorithm.

3.4.2 Functional Gradient Boosting

sec:FGD

In the nonparametric approach from the previous subsection, we are only looking at the observed data points, and not at neighboring points in $\mathcal{X}$ space. We have to keep in mind that although we are optimizing the empirical risk over a specific data set, we are actually trying to minimize the expected value (3.5), i.e.,

$$\text{Err}(\theta) = \mathbb{E}_{Y,X}\left[\rho(Y, \theta(\boldsymbol{X}))\right],$$

over all possible values of $\boldsymbol{X}$ and Y in the joint distribution. In addition, we wish to have an interpretable model. Therefore, the we must impose smoothness to neighboring points in the $\mathcal{X}$ space. We can do this by choosing steps of (parameterized) *functions* instead of steps of function *values*, which is what we did in the previous subsection. Therefore, since the solutions are parameterized functions, and we are performing gradient descent, the approach by Friedman (2001) develops a *functional* gradient descent (FGD) algorithm, a gradient

descent search in parameter space. We now consider the estimate $\hat{\theta}$ to be a function which takes values in $\mathcal{X}$,

$$\hat{\theta}(\boldsymbol{x}) = \beta_0 + \sum_{m=1}^{m_{\text{stop}}} f^{[m]}(\boldsymbol{x}),$$

and it is composed of sums of an initial value β_0, and steps $\left\{f^{[m]}\right\}_{m=1}^{m_{\text{stop}}}$. What is new is that now each $f^{[m]}(\boldsymbol{x})$ is a parameterized function which we will estimate. Since f is now a parameterized function, we do not consider the training error $\overline{\text{err}}(\theta)$ to be a function of N individual function parameter values $\theta(\boldsymbol{x}_i)$, $i = 1, \ldots, N$, but rather just a function of θ. Hence, to derive the negative gradient of an estimate $\hat{\theta}$, i.e., we must differentiate the loss function ρ with respect to θ instead of $\theta(x_i)$. The generalized residuals, or the negative gradient, then become

$$\boldsymbol{u} = \left(-\frac{\partial}{\partial\theta}\rho(y_i, \hat{\theta}(x_i))\right)_{i=1}^{N}.$$

Then we iterate, let us say, at each step $m > 0$ first calculating the generalized residuals of the previous iteration,

$$\boldsymbol{u}^{[m-1]} = \left(-\frac{\partial}{\partial\theta}\rho(y_i, \hat{\theta}^{[m-1]}(x_i))\right)_{i=1}^{N},$$

like we have seen before. Here we insert the model from the previous step, $\hat{\theta}^{[m-1]}$. We now perform a gradient descent step. However, we now constrain ourselves to steps which are functions of a base learner

$$\mathcal{H}(\cdot),$$

to ensure smoothness and generalizability, as discussed above.

A base learner is a class of functions $\mathcal{H}$ which is regularized, and often a simple effect of a parameter β. Typical examples are linear least squares, stumps (trees with one split, see Bühlmann and Hothorn, 2007; Hastie et al., 2009), and splines with a few degrees of freedom. There are several reasons to use simple base learners in each step. One is that there often exists fast methods for estimating a single base learner. Therefore there will be little computational cost in each step. Secondly, there is more to gain by combining simple learners, rather than combining complex learners. If we want to use complex learners, it is better to use another algorithm.

To take the steepest gradient in a functional sense, we must choose the realization of the base learner class $\mathcal{H}$ that produces the function $\hat{h}^{[m]}$ that is *most parallel* to $\boldsymbol{u}^{[m-1]}$. Another way of looking at it is that the best update is the member of the base learner function class h that is most correlated with $\boldsymbol{u}^{[m-1]}$ over the data distribution. This it means that $\hat{h}^{[m]}$ is the best approximation of the generalized residuals $\boldsymbol{u}^{[m-1]}$ by using $h(\cdot)$ to approximate with. Equivalently, $\hat{h}^{[m]}$ is the projection of the generalized residuals onto the space spanned by the base learner function class. We obtain $\hat{h}_m$ by fitting the base learner $\mathcal{H}$ to the generalized residuals. The specific method of fitting will depend on the base learner. If, for example, the base learner is a linear regressor, then the base learner will be

$$\hat{h}^{[m]} = \left(\hat{\boldsymbol{\beta}}^{[m]}\right)^T \boldsymbol{u}^{[m-1]},$$

where the parameter is found by

$$\hat{\boldsymbol{\beta}}^{[m]} = \left(\boldsymbol{x}^T\boldsymbol{x}\right)^{-1}\boldsymbol{x}^T\boldsymbol{u}^{[m-1]}$$

Having estimated the base learner, we do a line search to find the appropriate step length to use in order to minimize the loss function the most,

$$a^{[m]} = \underset{a}{\operatorname{argmin}}\, \overline{\text{err}}\left(\hat{\theta}^{[m-1]} + a \cdot \hat{h}^{[m]}.\right).$$

We add the estimated learner times the step length to the current model, obtaining

$$\hat{\theta}^{[m]}(\cdot) \leftarrow \hat{\theta}^{[m-1]}(\cdot) + a^{[m]}\hat{h}^{[m]}(\cdot).$$

We iterate this procedure until a stopping criterion is met. The convention that has emerged is to specify a number of iterations m_{stop}. This is the most important tuning parameter, and we will discuss it in the next subsection. The resulting model

$$\hat{\theta}_{\text{FGD}}(\cdot) = \hat{\theta}^{[m_{\text{stop}}]}(\cdot)$$

has the additive structure that we discussed earlier, namely

$$\hat{\theta}(\boldsymbol{x}) = \beta_0 + \sum_{m=1}^{m_{\text{stop}}} f^{[m]}(\boldsymbol{x}),$$

where each

$$f^{[m]}(\cdot) = a^{[m]} \cdot \hat{h}^{[m]}(\cdot).$$

This structure is a direct effect of the gradient descent algorithm, as the aggregation of base learners is strictly additive: In every iteration, small increments are added to the additive predictor. For a schematic overview of this algorithm, see Algorithm 2.

The algorithm just described calculates error terms in each iteration, and performs functional gradient descent on them. It is a very general framework, and it only requires a data set, a differentiable loss function, and a base learner. It is therefore quite straightforward to derive specific algorithms to use for specific models: It is just a matter of plugging in a chosen loss function and deriving its negative gradient. This gives great flexibility. We will now discuss the tuning parameters of the algorithm, which are very important to achieve good performance.

3.4.3 Tuning parameters

3.4.3.1 Step length

In the original generic functional gradient boosting algorithm, Algorithm 2, the step length a_m for each iteration is found through a line search, as in gradient descent. Friedman (2001) says that fitting the data too closely may be counterproductive, and result in overfitting. This has indeed proven to be true. To avoid overfitting, we must constrain the fitting procedure. This constraint is called regularization. Friedman therefore proposes to regularize each step in the algorithm by a common learning rate, $\nu \in (0, 1]$. As we will see, most modern boosting algorithms omit the step of the line search entirely, i.e. step

Algorithm 2 Gradient boosting, or, generic Functional Gradient Descent (FGD)

algo:fgd

1. Start with a data set $D = \{x_i, y_i\}_{i=1}^N$ and a chosen loss function $\rho(y, \theta(x))$, for which we wish to minimize the empirical risk, i.e., the loss function evaluated on the samples,

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, \overline{\operatorname{err}}(\theta) = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} \rho(y_i, \theta(x_i)).$$

2. Set iteration counter m to 0. Initialize the additive predictor by setting $\hat{f}_0(\cdot)$ to a constant β_0. This constant should be the maximizer of the loss function,

$$\beta_0(\cdot) = \underset{c}{\operatorname{argmin}}\, \overline{\operatorname{err}}(c),$$

and it can be found e.g. through numerical maximization.

3. Specify a base learner class h, e.g. linear least squares.

algo-fgd-step-inc

4. Increase m by 1.

5. Compute the generalized residuals (the negative gradient vector) of the previous iteration,

$$\boldsymbol{u}^{[m-1]} = \left(-\frac{\partial}{\partial \theta} \rho(y_i, \hat{\theta}^{[m-1]}(x_i)) \right)_{i=1}^{N}$$

6. Fit base learner h to the generalized residuals $\boldsymbol{u}$ to obtain a fitted version $\hat{h}^{[m]}$.

algo-fgd-step-line

7. Find best step length for $a^{[m]}$ by a line search:

$$a^{[m]} = \underset{a}{\operatorname{argmin}}\, \overline{\operatorname{err}} \left(\hat{\theta}^{[m-1]} + a \cdot \hat{h}^{[m]}. \right).$$

algo-fgd-step-last-loop

8. Update the current estimated θ,

$$\hat{\theta}^{[m]}(\cdot) \leftarrow \hat{\theta}^{[m-1]}(\cdot) + a^{[m]} \cdot \hat{h}^{[m]}(\cdot).$$

9. Repeat steps 4 to 8 (inclusive) until the iteration number m is m_{stop}.

10. Finally, return the estimated

$$\hat{\theta}_{\text{FGD}}(\cdot) = \hat{\theta}^{[m_{\text{stop}}]}(\cdot) = \beta_0 + \sum_{m=1}^{m_{\text{stop}}} a^{[m]} \hat{h}^{[m]}(\cdot).$$

7 in Algorithm 2. Instead, they fix a learning rate, or more commonly, step length ν. The choice of this step length is not of critical importance as long as it is sufficiently small (Schmid and Hothorn, 2008), i.e., it produces sufficient shrinkage, but the convention is to use $\nu = 0.1$ (Mayr et al., 2014a). This reduces the complexity of the algorithm, and it reduces the number of tuning parameters to the number of iterations m_{stop} only. There is a tradeoff between the number of iterations m_{stop} and the size of the step length ν. If the step length is smaller, then a larger number of iterations is needed, and conversely, if the step length is large, a smaller number of iterations is needed. We will now discuss the number of iterations.

subsec: iterations

3.4.3.2 Number of iterations

With a fixed step length (learning rate), the main tuning parameter for gradient boosting is the number of iterations m_{stop}, i.e. the number of steps performed before the algorithm is stopped. Its value is critical: If m_{stop} is too small, the model will underfit and it cannot fully incorporate the influence of the effects on the response and will consequently have poor performance. On the other hand, too many iterations will result in overfitting, leading to poor generalization. We know that we have either overfitting or underfitting if the estimated model $\hat{\theta}$ causes a high value of the test error $\text{Err}(\hat{\theta})$. It is easiest to notice overfitting if we calculate the test error as a function of the model complexity in. In boosting, the model complexity increases with the number of steps. The normal behaviour is that the test error will first decrease for a number of iterations, but then it will start to increase again. The training error $\overline{\text{err}}$, on the other hand, will continue to decrease, so it is important to monitor by calculating test error. The number of iterations is a very important tuning parameter. We will discuss it more in-depth in Section 3.6.

3.5 High dimensions and component-wise gradient boosting

sec:component

3.5.1 Problems in high dimensions

In modern biomedical statistics, it is crucial to be able to handle high-dimensional data. In some situations, a data set consists of more predictors p than observations N. When p is much larger than N ($p \gg N$), we talk about high-dimensional settings. In order to address the issue of analyzing high-dimensional data sets, a variety of regression techniques have been developed over the past years. Many of these techniques are characterized by a built-in mechanism for regularization. One such technique, which can cope with the $p \gg N$ situation, is a version of boosting called component-wise boosting.

subsec:comp-wise approach

3.5.2 The component-wise boosting approach

The key idea of the component-wise approach to gradient boosting is to add the effect of only one variable at a time, instead of adding a small effect from all variables, as is the case in the generic FGD algorithm (Algorithm 2). Component-wise gradient boosting is an algorithm which works very well in these settings. In fact, Buhlmann believes that it is mainly in the case of

high-dimensional predictors that boosting has a substantial advantage over classical approaches (Bühlmann, 2006). The component-wise approach was first proposed in (Bühlmann and Yu, 2003), and component-wise boosting is a very active field of research (Bühlmann, 2006; Mayr et al., 2014a,b, 2017). Consider yet again the case where we have a data set $D = (\boldsymbol{x}_i, y_i)_{i=1}^N$, where $\boldsymbol{x}_i \in \mathbb{R}^p$ are covariate vectors of a high dimension p, and N is the number of observations. Specifically, we are in a setting where $p > N$. We want to finding the parameter θ which minimizes the empirical risk of a chosen loss function ρ on the data set,

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, \overline{\operatorname{err}}(\theta) = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} \rho(\boldsymbol{y}_i, \theta(\boldsymbol{x}_i)),$$

where the parameter θ is a predictor $\theta \colon \mathbb{R}^p \to \mathbb{R}$. However, we will now let θ be an additive predictor, meaning it is a sum of partial effects of covariates,

$$\theta(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{p} f_j(x_j).$$

These partial effects of the covariates will be estimated by component-wise learners in an iterative fashion.

3.5.3 The component-wise boosting algorithm

The structure of the component-wise boosting algorithm is very much the same as the generic functional gradient boosting algorithm (Algorithm 2), but with some additional steps. As mentioned, instead of using a base learner which incorporates all predictors, we use a set $\mathcal{H}$ of base learners consisting of a separate base learner for each component of the covariates. The learners typically share the same structure, i.e., they are the same base learner, applied to different covariates. For example, if we use a linear least squares model as base learners, the set of base learners would be

$$\mathcal{H} = \{h_1(\boldsymbol{x}; \beta_j)\}_{j=1}^p = \{\beta_j x_j\}_{j=1}^p$$

It is not necessarily the case that the learners have the same structure, it is also possible to include e.g. spline learners for some components, and least squares for others. However, in the following, we will assume that each component only has one base learner, and that it is the same structure, which simplifies our notation a bit.

The initialization of the algorithm is the same as in the FGD algorithm: We first initialize the additive predictor $\hat{\theta}^{[0]}$ to a constant β_0. We then iterate. In a given iteration m, we first construct the generalized residuals, by calculating the negative gradient. Here we insert the additive predictor from the previous step, namely $\hat{\theta}^{[m-1]}$,

$$\boldsymbol{u}^{[m-1]} = \left(-\frac{\partial}{\partial \theta} \rho(y_i, \hat{\theta}(\boldsymbol{x}_i) \right)_{i=1}^{N}.$$

Note that this calculation is exactly like in the generic gradient boosting algorithm. While the generic FGD algorithm here only estimated a single base

learner, in the component-wise we estimate all base learners separately. We obtain p estimated functions

$$\hat{h}_1^{[m]}(\cdot),\ \hat{h}_2^{[m]}(\cdot),\ \ldots,\ \hat{h}_p^{[m]}(\cdot).$$

These estimated functions can again be viewed as approximations of the negative gradient vector, or equivalently the projection of the negative gradient vector onto the space spanned by the component-wise base learner. However, these are projections onto only one dimension of the covariate space. We wish to reduce the error $\overline{\text{err}}$ as much as possible, and so we select the covariate which has a corresponding estimated base learner which explains as much as possible of the variation in $\boldsymbol{u}^{[m-1]}$. To select the best-fitting base-learner, we select the one with the smallest residual sum of squares error

$$j^{[m]} = \underset{j\in\{1,2,\ldots,p\}}{\operatorname{argmin}} \sum_{i=1}^{N} \left(u_i - \hat{h}_j^{[m]}\right)^2.$$

Note that this makes sense from a linear algebra perspective: Choosing the one with minimal RSS means that we choose the one with the smallest projection error, or the one with the most signal. We add this best-fitting base-learner $h_{j^{[m]}}^{[m]}$ to the current model, with a pre-specified step length of ν, again typically set to 0.1, for the purpose of regularization. Hence the model after iteration m is

$$\hat{\theta}^{[m]}(\cdot) \leftarrow \hat{\theta}^{[m-1]}(\cdot) + \nu \cdot \hat{h}_{j^{[m]}}^{[m]}.$$

Seen from a component-wise perspective, we update the predictor of the selected component,

$$\hat{f}_{j^{[m]}}^{[m]}(\cdot) \leftarrow \hat{f}_{j^{[m]}}^{[m-1]}(\cdot) + \nu \cdot \hat{h}_{j^{[m]}}^{[m]},$$

This should be better!

and for all other components $j \in \{j\colon j \neq j^{[m]},\ j = 1, 2, \ldots, p\}$, the update in iteration m is simply to keep the predictor from the last iteration

$$\hat{f}_j^{[m]}(\cdot) \leftarrow \hat{f}_j^{[m-1]}.$$

We continue iterating until the iteration number m reaches the pre-specified stopping iteration m_{stop}. We will discuss selection of m_{stop} in Section 3.6. The final additive predictor becomes

$$\hat{\theta} = \hat{\theta}^{[m_{\text{stop}}]} = \beta_0 + \sum_{m=1}^{m_{\text{stop}}} \nu \cdot \hat{h}_{j^{[m]}}^{[m]}(\cdot).$$

Note that any base-learner h_j can be selected at multiple iterations. The partial effect of the variable x_j is the sum of the estimated corresponding base learner in all iterations where it was selected. Hence each $\hat{\theta}_j^{[m_{\text{stop}}]}(x_j)$ can be seen as i.e.,

$$\hat{\theta}_j(x_j) = \sum_{m=1}^{m_{\text{stop}}} \nu \cdot \hat{h}_j^{[m]}(x_j) I\left(j^{[m]} = j\right),$$

where $I(\cdot)$ is an indicator function. Hence the resulting additive predictor is a sum of component-wise predictors in the GAM form of

$$\hat{\eta}(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{p} \hat{f}_j(x_j).$$

For a schematic overview of the algorithm, see Algorithm 3. This algorithm has been implemented in R as `mboost` (Hothorn et al., 2018; Hofner et al., 2014; Hothorn et al., 2010), and it contains many different base learners which can be included in the model.

3.5.4 Component-wise boosting performs data-driven variable selection

sec:variable-selection

Stopping the algorithm before every base-learner was at least selected once effectively excludes all non-selected base-learners, and thus also the corresponding covariates, from the final model. The algorithm is therefore able to perform variable selection and model fitting simultaneously. Furthermore, early stopping shrinks effect estimates toward zero (Bühlmann and Hothorn, 2007; De Bin, 2016), similar to L_1-penalized regression such as the lasso (Tibshirani, 1996; Efron et al., 2004). Shrinkage of effect estimates lead to a lower variance and therefore to more stable and accurate predictions (Efron, 1975; Copas, 1983; Hastie et al., 2009).

In other words, if the number of iterations m_{stop} is small enough, the component-wise gradient boosting algorithm will carry out automatic variable selection. In particular the base learners applied to irrelevant variables will never be considered in the updating step, and therefore many of the covariates in $\boldsymbol{x}$ will not be a part of the final model. Some predictors will have more explanatory power, or signal, than others, and so they will be selected more often. This is because some predictors are more correlated with the output than others.

3.6 Selecting the iteration number m_{stop}

sec:stop

As we have mentioned in subsection 3.4.3.2, the crucial tuning parameter in boosting is the number of iterations, m_{stop}. Stopping a boosting procedure early enough will lead to variable selection and shrinks the parameter estimates toward zero. In the case of $p < N$, with $m \to \infty$, the parameters in boosting will converge towards the maximum likelihood estimates (De Bin, 2016), i.e., minimizing the in-sample error. We are, on the other hand, interested in minimizing the test error $\text{Err}(\theta)$ with respect to θ. When using a gradient boosting algorithm, we do so on a data set D. We choose an appropriate loss function, we specify base learners, and we specify a learning rate. When these four factors are fixed, the parameter path will be deterministic. This means that running the boosting algorithm the same number of steps m several times will lead to the same $\hat{\theta}^{[m]}$. There is no randomness in the estimates. This is quite obvious if we think about it, because at any given step, the algorithm selects the component and learner which leads to the best decrease in training error, and there is nothing random in that. The point of this is that we can

algo:component-wise

Algorithm 3 Component-wise gradient boosting

1. Start with a data set $D = \{x_i, y_i\}_{i=1}^N$ and a chosen loss function $\rho(y, \theta(x))$, for which we wish to minimize the empirical risk, i.e., the loss function evaluated on the samples,

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, \overline{\operatorname{err}}(\theta) = \underset{\theta}{\operatorname{argmin}}\, \frac{1}{N} \sum_{i=1}^{N} \rho(y_i, \theta(x_i)).$$

2. Set iteration counter m to 0. Specify a step length ν. Initialize the additive predictor by setting $\hat{f}_0(\cdot)$ to a constant β_0. This constant should be the maximizer of the loss function,

$$\beta_0(\cdot) = \underset{c}{\operatorname{argmin}}\, \overline{\operatorname{err}}(c),$$

and it can be found e.g. through numerical maximization.

3. Specify a set of base learners $\mathcal{H} = \{h_1(\cdot), \ldots, h_p(\cdot)\}$, where each h_j is univariate and takes column j of $\boldsymbol{X}$.

first-step

4. Increase m by 1.

5. Compute the negative gradient vector, i.e., the generalized residuals after the previous iteration of the boosted model,

$$\boldsymbol{u}^{[m-1]} = \left(-\frac{\partial}{\partial \theta} \rho(y_i, \hat{\theta}(x_i)) \right)_{i=1}^{N}.$$

6. For each base learner $h_j \in \mathcal{H}, j = 1, \ldots, p$, estimate $\hat{h}_j^{[m]}$ by fitting $(\boldsymbol{x}_i, u_i)_{i=1}^N$ using the base learner $h_j(\cdot)$. We obtain

$$\hat{h}_1^{[m]}(\cdot), \hat{h}_2^{[m]}(\cdot), \ldots, \hat{h}_p^{[m]}(\cdot).$$

7. Select the best-fitting component $j^{[m]}$, i.e., with lowest RSS,

$$j^{[m]} = \underset{j \in \{1,2,\ldots,p\}}{\operatorname{argmin}} \sum_{i=1}^{N} \left(u_i - \hat{h}_j^{[m]} \right)^2.$$

last-step

8. Update the current model with the best-fitting model from the current iteration

$$\hat{\theta}^{[m]}(\cdot) \leftarrow \hat{\theta}^{[m-1]}(\cdot) + \nu \cdot \hat{h}_{j^{[m]}}^{[m]}(\cdot).$$

9. Repeat steps 4 to 8 (inclusive) until the iteration number m is m_{stop}.

10. Return the final boosted additive predictor

$$\hat{\theta}(\cdot) = \hat{\theta}^{[m_{\text{stop}}]}(\cdot) = \beta_0 + \sum_{m=1}^{m_{\text{stop}}} \nu \cdot \hat{h}_{j^{[m]}}^{[m]}(\cdot)$$

view the test error of an estimated boosting model $\hat{\theta}^{[m]}$ as a function of the number of iterations used to produce it, i.e.,

$$\text{Err}(m) = \text{Err}(\hat{\theta}^{[m]}). \tag{3.7}$$

{test-error-m}

There exists an m which minimizes $\text{Err}(m)$. Many authors state that the algorithm should be stopped early, but do not go further into the details. What we want is therefore a good method for approximating $\text{Err}(m)$. This can be done in a number of ways. Common model selection criteria such as the Akaike Information Criteria (AIC) may be used, however the AIC is dependent on estimates of the model's degrees of freedom. Initial versions of boosting used it, but practice showed that AIC-based stopping rules lead to overfitting (Mayr et al., 2012b). For L_2Boost, Bühlmann and Hothorn (2007) suggest that $\text{df}(m) = \text{trace}(B_m)$ is a good approximation. Here B_m is the hat matrix resulting from the boosting algorithm. This was, however, shown by Hastie (2007) to always underestimate the actual degrees of freedom. Mayr et al. (2012b) propose a sequential stopping rule using subsampling. However this is computationally very expensive and not really used in practice. None of these rules, in other words, are optimal to use. Instead, cross-validation, a general and very common method for selection of tuning parameters in statistics, is what is used in almost all cases (Mayr et al., 2014a,b, 2017). Cross-validation is flexible and easy to implement. It is somewhat computationally demanding, because it requires several full runs of the boosting algorithm, but it is otherwise quite simple. We now give an explanation of this procedure.

subsec:K-fold

3.6.1 K-fold cross-validation

K-fold cross-validation (Lachenbruch and Mickey, 1968), or simply cross-validation, is a general method for estimating the test error and therefore commonly used for selection of penalty or tuning parameters. In cross-validation, the data are randomly split into K rougly equally sized folds. For a given fold k, all folds except k act as the training data and used to estimate the model. The resulting model is then evaluated on the unseen data, namely the observations belonging to fold k. This procedure is repeated for all $k = 1, \ldots, K$. An estimate of the test error is obtained by averaging over the test errors evaluated in each left-out fold. Let $\kappa(k)$ be the set of indices for fold k. The cross-validated estimate for a given m then becomes

$$\text{CV}(m) = \frac{1}{K} \sum_{k=1}^{K} \sum_{i \in \kappa(k)} \rho(y_i, \hat{y}_i^{-\kappa(k)}).$$

$\text{CV}(m)$ is an estimate of the test error of gradient boosting as a function of the iteration number m (3.7). For each iteration m, we calculate the estimate of the cross-validated prediction error $\text{CV}(m)$. We choose m_{stop} to be the minimizer of this error,

$$m_{\text{stop}} = \underset{m}{\text{argmin}}\, \text{CV}(m).$$

Typical values for K are 5 or 10, but in theory one can choose any number. The extreme case is $K = N$, called leave-one-out cross-validation, where all but one observation is used for training and the model is evaluated on the observation that was left out. In this case, the outcome is deterministic, since there is no randomness when dividing into folds.

3.6.2 Stratified cross-validation

When dividing an already small number of survival data observations into K folds, we might risk getting folds without any observed deaths, or in any case, very few. In stratified cross validation, we do not divide the folds entirely at random, but rather, try to divide the data such that there is an equal amount of uncensored data in each fold. As before, let $\kappa(k)$ be the set of indices for fold k. Divide the observed data into K folds, as with usual cross validation, to get an index set $\kappa_{\delta=1}(k)$ for a given k. Similarly, divide the censored data into K folds, obtaining $\kappa_{\delta=0}(k)$. Finally, $\kappa(k)$ is the union of these sets: $\kappa(k) = \kappa_{\delta=1}(k) \cup \kappa_{\delta=0}(k)$. For a detailed description of 10-fold cross-validation issues in the presence of censored data, see Kohavi (1995).

subsec:repeated-cv

3.6.3 Repeated cross-validation

The randomness inherent in the cross-validation splits has an effect on the resulting m_{stop}. This is true in general, but it is especially true for real-life survival data, because such data sets typically have small effective sample sizes (number of observed events). We can easily imagine that we can end up with quite different values for m_{stop} for two different splits of the data, depending on which folds the events end up in. It has been very effectively demonstrated that the split of the folds has a large impact on the choice of m_{stop} (Seibold et al., 2018). To reduce the impact of this issue, Seibold et al. (2018) suggest to repeat the cross-validation scheme a few times and average the results. They show that repeating the cross-validation procedure even only 5 times effectively averages out the randomness. In other words, we divide the data into K folds, and repeat this J times. Now let $\kappa(j, k)$ be the k-th fold in the j-th split. We end up with a new estimate for the prediction error,

$$\text{RCV}(m) = \frac{1}{J}\sum_{j=1}^{J}\frac{1}{K}\sum_{k=1}^{K}\sum_{i\in\kappa(j,k)}\rho(y_i, \hat{y}_i^{-\kappa(j,k)}).$$

Again, $\text{RCV}(m)$ is also an estimate of (3.7), but with less variance due to averaging several results. As before, we choose m_{stop} to be the minimizer of this error,

$$m_{\text{stop}} = \operatorname*{argmin}_m \text{RCV}(m).$$

To carry out this in practice when using a boosting algorithm, we let the boosting algorithm run for $m = 1$ to $m = M$, where M is a large number that we are sure will result in a overfitted model. Thus we ensure that we find the minimizing m, and not a local minimum.

3.7 Multidimensional boosting

A limitation of the boosting methods we have described so far, as well as of L_1-penalized estimation such as the lasso method (Tibshirani, 1996), is that they are designed for statistical problems involving a one-dimensional prediction function. By only considering such functions, we are restricted to estimating models which only model a single quantity of interest, which is almost always the mean. In many applications, modelling only one parameter will not be sufficient

(Kneib, 2013). We want to be able to estimate more general models, in which more quantities, e.g. the drift and the threshold of the models described in section 2.3, can be explained by covariates. To properly estimate such an FHT model, we therefore need a more general gradient boosting algorithm which is able to estimate several parameters. Typical examples of multidimensional estimation problems are classification with multiple outcome categories and regression models for count data. Another example is estimating models in the GAMLSS family (Rigby and Stasinopoulos, 2005). GAMLSS, which refer to "generalized additive models for location, scale and shape," are a family of models that relates not only the mean, but all parameters of the outcome distribution to the available covariates. GAMLSS are in this sense an extension of GAM models (Hastie and Tibshirani, 1990). A gradient boosting algorithm called *gamboostLSS* was developed for boosting such models (Mayr et al., 2012a). The algorithm framework used in *gamboostLSS* is inspired by the multidimensional boosting algorithm first introduced in Schmid et al. (2010). We will here explain the *gamboostLSS* algorithm, as presented in Mayr et al. (2012a).

There are two versions of the multidimensional gradient boosting algorithm. One, which originated first, is now referred to as cyclical, and the other is the noncyclical version. In the cyclical version, each parameter dimension is updated with a component-wise learner. In the noncyclical version, only one parameter dimension is updated, specifically the parameter dimension update that leads to the best decrease in training error (log-likelihood). The cyclical version is somewhat more stable in the variable selection, but it requires a vector of stopping iterations $\boldsymbol{m}_{\text{stop}}$, to be found by grid search (Thomas et al., 2018). Thus, it is much faster to use the noncyclical version.

3.8 Cyclical *gamboostLSS*

sec:gamlssboost

A key feature of the GAMLSS model family is that every parameter of the conditional response distribution is modelled by its own predictor and associated link function. Traditional GAMs (Hastie and Tibshirani, 1990) are typically restricted to modelling the conditional mean of the response variable, and treats possible other distributional parameters as fixed. GAMLSS, on the other hand, allows for regression of each distribution parameter on the covariates. Common distribution parameters are location, scale, skewness and kurtosis, but degrees of freedom (of a t-distribution) and zero inflation probabilities can be modelled as well (Mayr et al., 2012a; Hofner et al., 2018). Thus, in the GAMLSS approach, the full conditional distribution of a multiparameter model is related to a set of predictor variables of interest. Similarly to in GAMs, in GAMLSS the structure of each predictor is assumed to be additive. Hence a wide variety of functional predictors can be included in each predictor. Examples include non-parametric terms based on penalized splines, varying-coefficient terms and spatial and subject-specific terms for repeated measurements. The estimation of GAMLSS coefficients is usually based on penalized likelihood maximization; for details on fitting procedures, see Rigby and Stasinopoulos (2005).

3.8.1 GAMLSS

subsec:GAMLSS

The GAMLSS model class assumes observations y_i for $i = 1, 2, \ldots, N$ that are conditonally independent given a set of covariates and after having accounted

for spatiotemporal effects. The conditional density

$$\psi(y_i|\boldsymbol{\theta}(x_i)), \tag{3.8}$$

{gamlss-density}

may depend on K distribution parameters

$$\boldsymbol{\theta}_i = (\theta_{i,1}, \theta_{i,2}, \ldots, \theta_{i,K})^T .$$

Each distribution parameter $\theta_k, k = 1, 2, \ldots, K$ is modelled by its own additive predictor η_k and depends additively on the covariates. θ_k is linked to its predictor by a known monotonic link function

$$g_k(\cdot).$$

Letting p_k be the number of covariates to be used for distribution parameter θ_k,

$$x_{k,1}, x_{k,2}, \ldots, x_{k,p_k}$$

are the covariates in the model of the parameter θ_k. A GAMLSS is given by the equations

$$\eta_k \coloneqq g_k(\theta_k) = \beta_{k,0} + \sum_{j=1}^{p_k} f_{k,j}(x_{k,j}),$$

for all $k = 1, 2, \ldots, K$. Here $\beta_{k,0}$ is the intercept for distribution parameter θ_k, and $f_{k,j}$ represents the type of effect that covariate j has on the distribution parameter θ_k, through the link function. In the case of a simple linear regression learner, a component-wise effect of component j on distribution parameter θ_k would be

$$f_{k,j}(x_{k,j}) = x_{k,j}\beta_{k,j},$$

where $\beta_{k,j}$ is a parameter to be estimated. Finally, η_k is the additive predictor for θ_k. Note that a GAMLSS reduces to a GAM (Hastie and Tibshirani, 1990) in the case where the distribution parameter vector is a scalar

$$\boldsymbol{\theta}_i = \theta_i = \mu_i,$$

i.e., the conditional mean of observation i. For parametric models, the unknown quantities of a GAMLSS can be estimated by maximizing the log-likelihood of an observed data set of N observations of the conditional density (3.8). The log-likelihood contribution for one sample is

$$\log\{\psi(\boldsymbol{\theta}(x_i)|y_i)\},$$

$\hat{\theta}$ will be a functional which works on the covariate vector $\boldsymbol{x}$. Hence the total log-likelihood is

$$l(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log\left(\psi(\boldsymbol{\theta}(x_i)|y_i)\right),$$

Denoting estimates of the prediction functions as $\hat{\eta}_k$, estimates of the distribution parameters $\boldsymbol{\theta}$ are then obtained from transforming back via the inverse link functions,

$$\hat{\theta}_k = g_k^{-1}(\hat{\eta}_{\theta_k}),$$

for all $k = 1, 2, \ldots, K$, i.e., $\hat{\boldsymbol{\theta}} = (\theta_1, \theta_2, \ldots, \theta_K)$ is an estimate of $\boldsymbol{\theta}$. After the original GAMLSS paper (Rigby and Stasinopoulos, 2005), a penalized likelihood approach based on modified versions of the backfitting algorithm for GAM estimation was developed by the same authors (Stasinopoulos and Rigby, 2007). Later, however, a gradient boosting algorithm, called *gamboostLSS*, was developed (Mayr et al., 2012a). The algorithm *gamboostLSS* uses a strategy for multidimensional boosting proposed by Schmid et al. (2010). Thomas et al. (2018) later coined the term "cyclical" to describe it and here we use this notation.

3.8.2 The *gamboostLSS* algorithm

The main idea of the cyclical multidimensional boosting algorithm is to have a boosting step for each parameter k in each iteration, and to successively update the predictors in each iteration. We cycle through all parameter dimensions in each boosting iteration. In every dimension k, we carry out one boosting iteration. This boosting iteration can in principle be the same as in the generic FGD algorithm (2), i.e., to estimate a full base learner which incorporates all covarietes. The *gamboostLSS* algorithm (Mayr et al., 2012a), however, uses the component-wise base learner strategy, introduced in section 3.5. This allows for model fitting in high-dimensional contexts. To use the gradient boosting approach for a multidimensional prediction function, we need to have existing partial derivatives of the loss function with respect to each predictor. Again since we are doing a gradient descent step, we as usual use the *negative* derivative. These negative partial derivatives are

$$-\frac{\partial}{\partial \eta_k}\rho(y_i, \boldsymbol{\eta}(x_i)) = \frac{\partial}{\partial \eta_k}\log(\psi(y_i|\boldsymbol{\theta}(x_i))),$$

for all $k = 1, 2, \ldots, K$. As in previous algorithms, we use these negative derivatives to construct generalized residual vectors. In this multidimensional approach, we now have K partial derivatives, and so we construct K different generalized residual vectors $\boldsymbol{u}_k$, $k = 1, 2 \ldots, K$. For a specific distribution parameter θ_k, we construct a generalized residual vector by computing the negative derivative with respect to each additive predictor η_k, and inserting the current estimate $\hat{\boldsymbol{\eta}}$, evaluated at each observation $(x_i, y_i)_{i=1}^N$. This yields

$$\begin{aligned}\boldsymbol{u}_k &= (u_{k,1}, u_{k,2}, \ldots, u_{k,N}) \\ &= \left(-\frac{\partial}{\partial \eta_k}\rho(y_i, \hat{\boldsymbol{\eta}}(x_i))\right)_{i=1}^N.\end{aligned}$$

Like in other gradient boosting algorithms, we need base learners. We specify component-wise base learners for *each* distribution parameter. In principle, these might be different, but one usually chooses the same type for all, only letting the base learners differ in which component and which parameter they affect. In other words, in general we have a set of base learners

$$\mathcal{H}_k = \{h_{k,1}, h_{k,2} \ldots, h_{k,p_1}\}$$

for each $k = 1, 2, \ldots, K$. As before, we use the base learners to estimate possible updates of the model based on single predictors, and we choose the one that improves the model the most.

The initialization of the algorithm is done analogously to the regular boosting method, by setting each parameter to a constant. These constants should be those that jointly maximize the log-likelihood. We find the optimal constant c_k for each distribution parameter, and then initialize the estimate of each predictor η_k as

$$\hat{\eta}_k^{[0]} = \beta_{k,0} = c_k,$$

for each $k = 1, 2, \ldots, K$. In the k-th step of iteration m, i.e. after having cycled through until component k, the estimated vector of additive predictors is denoted $\hat{\boldsymbol{\eta}}_{k-1}^{[m]}$, and it is

$$\hat{\boldsymbol{\eta}}_{k-1}^{[m]} = \left(\hat{\eta}_1^{[m]}, \hat{\eta}_2^{[m]}, \ldots, \hat{\eta}_{k-1}^{[m]}, \hat{\eta}_k^{[m-1]}, \hat{\eta}_{k+1}^{[m-1]}, \ldots, \hat{\eta}_K^{[m-1]}\right).$$

Here the additive predictors of the preceding parameter dimensions, $1, 2, \ldots, k-1$, have been updated in the current iteration m. Hence they are denoted with the current iteration, as $\hat{\eta}_1^{[m]}$, $\hat{\eta}_2^{[m]}$, until $\hat{\eta}_{k-1}^{[m]}$. The following dimensions $k+1, \ldots, K$ have not been updated in iteration m, and hence they are denoted with iteration $m-1$:

$$\hat{\eta}_{k+1}^{[m-1]}, \hat{\eta}_{k+2}^{[m-1]}, \text{ until } \hat{\eta}_K^{[m-1]}.$$

We are in this step going to update dimension k, i.e., going from $\hat{\eta}_k^{[m-1]}$ to $\hat{\eta}_k^{[m]}$. We calculate a vector of generalized residuals for dimension k by calculating the k-th partial derivative and inserting the current estimated vector of additive predictors $\hat{\boldsymbol{\eta}}_{k-1}^{[m]}$, and evaluating it at the observations $x_1, x_2, \ldots, x_N$. This yields the generalized residual vector

$$\begin{aligned}
\boldsymbol{u}_k^{[m-1]} &= (u_{k,1}^{[m-1]}, u_{k,2}^{[m-1]}, \ldots, u_{k,N}^{[m-1]}) \\
&= \left(-\frac{\partial}{\partial \eta_k} \rho(\hat{\eta}_1^{[m]}(\boldsymbol{x}_i), \hat{\eta}_2^{[m]}(\boldsymbol{x}_i), \ldots, \hat{\eta}_{k-1}^{[m]}(\boldsymbol{x}_i), \hat{\eta}_{k+1}^{[m-1]}(\boldsymbol{x}_i), \ldots, \hat{\eta}_K^{[m-1]}(\boldsymbol{x}_i))\right)_{i=1}^{N} \\
&= \left(-\frac{\partial}{\partial \eta_k} \rho(y, \hat{\boldsymbol{\eta}}_{k-1}^{[m]}(\boldsymbol{x}_i))\right)_{i=1}^{N}.
\end{aligned}$$

Again, like in a regular component-wise boosting algorithm, we fit all component-wise base learners separately to this residual vector $\boldsymbol{u}_k^{[m-1]}$. Of these learners, select the best fitting component $j_k^{[m]}$ like previously, by selecting the estimated learner which fits best according to RSS,

$$j_k^{[m]} = \underset{j \in \{1,2,\ldots,p_k\}}{\operatorname{argmin}} \sum_{i=1}^{N} \left(u_{k,i}^{[m-1]} - \hat{h}_{k,j}^{[m]}\right)^2.$$

We update the additive predictor in dimension k by the usual

$$\hat{f}_k^{[m]} \leftarrow \hat{f}_k^{[m-1]} + \nu \cdot \hat{h}_{j_k^{[m]}}^{[m]}(\cdot).$$

A schematic representation of the updating process using this algorithm in a given iteration m follows:

$$\begin{aligned}
&\frac{\partial}{\eta_1}\rho\left(y, \eta_1^{[m-1]}, \eta_2^{[m-1]}, \eta_3^{[m-1]}, \ldots, \eta_{K-1}^{[m-1]}, \eta_K^{[m-1]}\right) \xrightarrow{\text{calculate}} \boldsymbol{u}_1^{[m-1]} \xrightarrow{\text{fit and update}} \hat{f}_1^{[m]} \\
&\quad\frac{\partial}{\eta_2}\rho\left(y, \hat{\eta}_1^{[m]}, \hat{\eta}_2^{[m-1]}, \hat{\eta}_3^{[m-1]}, \ldots, \hat{\eta}_{K-1}^{[m-1]}, \hat{\eta}_K^{[m-1]}\right) \xrightarrow{\text{calculate}} \boldsymbol{u}_2^{[m-1]} \xrightarrow{\text{fit and update}} \hat{f}_2^{[m]} \\
&\qquad\frac{\partial}{\eta_3}\rho\left(y, \hat{\eta}_1^{[m]}, \hat{\eta}_2^{[m]}, \hat{\eta}_3^{[m-1]}, \ldots, \hat{\eta}_{K-1}^{[m-1]}, \hat{\eta}_K^{[m-1]}\right) \xrightarrow{\text{calculate}} \boldsymbol{u}_3^{[m-1]} \xrightarrow{\text{fit and update}} \hat{f}_3^{[m]} \\
&\qquad\qquad\ldots \\
&\quad\frac{\partial}{\eta_{K-1}}\rho\left(y, \hat{\eta}_1^{[m]}, \hat{\eta}_2^{[m]}, \hat{\eta}_3^{[m]}, \ldots, \hat{\eta}_{K-1}^{[m-1]}, \hat{\eta}_K^{[m-1]}\right) \xrightarrow{\text{calculate}} \boldsymbol{u}_{K-1}^{[m-1]} \xrightarrow{\text{fit and update}} \hat{f}_{K-1}^{[m]} \\
&\qquad\frac{\partial}{\eta_K}\rho\left(y, \hat{\eta}_1^{[m]}, \hat{\eta}_2^{[m]}, \hat{\eta}_3^{[m]}, \ldots, \hat{\eta}_{K-1}^{[m]}, \hat{\eta}_K^{[m-1]}\right) \xrightarrow{\text{calculate}} \boldsymbol{u}_K^{[m-1]} \xrightarrow{\text{fit and update}} \hat{f}_K^{[m]}
\end{aligned}$$

For a schematic overview of this cyclical multidimensional boosting algorithm, see Algorithm 4. Note that this algorithm resembles the backfitting strategy of Hastie and Tibshirani (1986). In both backfitting and this multidimensional boosting strategy, components are updated successively by using estimates of the other components as offset values. In backfitting, a completely new estimate of f^* is determined in every iteration. In gradient boosting, however, the estimates are only slightly modified in each iteration, and the number of iterations is parameter specific, and is pre-specified.

3.8.3 Tuning parameters

The main tuning parameters in the cyclical multidimensional gradient boosting algorithm are the stopping iterations $\boldsymbol{m}_{\text{stop}} = m_{\text{stop},1}, \ldots, m_{\text{stop},K}$. As in the one-dimensional gradient boosting algorithm, we should not let the algorithm run until convergence, since that will lead to overfitting, and we want to have a low test error. We therefore need to find estimates of the stopping iterations by cross-validation (Schmid et al., 2010). However, to properly tune these parameters, it is necessary to perform a multidimensional search, which is usually done by implementing a so-called grid search. One divides the search space into a multidimensional grid, obtaining tuples of configurations. On each tuple, we should use cross-validation, as usual, and the next subsection explains the procedure.

3.8.4 Grid search cross-validation in gradient boosting

grid-search

To find a vector of length K of optimal iterations $\boldsymbol{m}_{\text{stop}} = m_{\text{stop},1}, \ldots, m_{\text{stop},K}$, we perform a K-dimensional grid search. We must first specify a minimum and maximum number of iterations for each parameter. Call these $M_{k,\min}$ and $M_{k,\max}$, respectively. We then divide this one-dimensional search space into a finite grid with N_k points, such that we obtain

$$M_{k,\min} = M_{k,1} < M_{k,2} < \ldots < M_{k,N_k-1} < M_{k,N_k} = M_{k,\max},$$

again for each $k = 1, 2, \ldots, K$. The total search space is the cartesian product of all of these grids. We illustrate with an example. Let $K = 3$, $M_{k,\min} = 1$,

algo:multi-cyclical

Algorithm 4 Multidimensional cyclical component-wise gradient boosting

1. Start with a data set $D = \{\boldsymbol{x}_i, y_i\}_{i=1}^N$ and a chosen loss function $\rho(y, \boldsymbol{\eta}(\boldsymbol{x}))$, for which we wish to minimize the empirical risk, i.e., the loss function evaluated on the samples,

$$\hat{\boldsymbol{\eta}} = \underset{\boldsymbol{\eta}}{\operatorname{argmin}}\, \overline{\operatorname{err}}(\boldsymbol{\eta}) = \underset{\boldsymbol{\eta}}{\operatorname{argmin}} \sum_{i=1}^{n} \rho(y_i, \boldsymbol{\eta}(\boldsymbol{x}_i)).$$

2. Initialize iteration counter m to 0. Initialize additive predictors to constants $\boldsymbol{\beta}_0 = (\beta_{1,0}, \beta_{2,0} \ldots, \beta_{k,0})$, e.g. to those that jointly maximize the training error,

$$\boldsymbol{\beta}_0 = \underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, \overline{\operatorname{err}}\,(\boldsymbol{\beta})\,.$$

initialization

3. Specify a set of base learners $\mathcal{H}_k$ for each predictor θ_k, for $k = 1, \ldots, K$. Specify a step length ν.

cyclic-proper-first

4. Increase m by 1.

5. Set k to 0.

cyclic-first

6. Increase k by 1.

7. If $m > m_{\text{stop},k}$, go to step 6. Otherwise compute the negative partial derivative $-\frac{\partial \rho}{\partial \eta_k}$ and evaluate at $\hat{\boldsymbol{\eta}}_{k-1}^{[m]}(x_i), i = 1, \ldots, N$, yielding the negative gradient vector

$$\boldsymbol{u}_k^{[m-1]} = \left(-\frac{\partial}{\partial \eta_k} \rho(y_i, \hat{\boldsymbol{\eta}}_{k-1}^{[m]}(x_i) \right)_{i=1}^{N}$$

8. Fit the negative gradient vector $\boldsymbol{u}_k^{[m-1]}$ to each of the p_k components of $\boldsymbol{X}$ separately, using each component's respective base learner. This yields p_k vectors of predicted values,

$$\hat{h}_{k,1}^{[m]}, \hat{h}_{k,2}^{[m]}, \ldots, \hat{h}_{k,p_k}^{[m]}.$$

9. Select the component of $\boldsymbol{x}$ with the best-fitting base learner, according to RSS,

$$j_k^{[m]} = \underset{j \in \{1,2,\ldots,p\}}{\operatorname{argmin}} \sum_{i=1}^{N} \left(u_{k,i} - \hat{h}_{k,j}^{[m]} \right)^2.$$

10. Update the predictor for parameter k in component $j^{[m]}$ by

$$\hat{\eta}_{k,j_k^{[m]}}^{[m]} \leftarrow \hat{\eta}_{k,j_k^{[m]}}^{[m-1]} + \nu \cdot \hat{h}_{j_k^{[m]}}^{[m]},$$

where ν is the real-valued step-length factor specified in step 3. For all other components, meaning each $j \in \{j \neq j_k^{[m]}, j = 1, 2, \ldots, p_k\}$, set the predictor to the one from the previous iteration,

$$\hat{\eta}_{k,j}^{[m]} \leftarrow \hat{\eta}_{k,j}^{[m-1]}.$$

cyclic-last

11. If $k < K$, go to step 6. If not, update the full model, $\hat{\boldsymbol{\eta}}^{[m]} \leftarrow \hat{\boldsymbol{\eta}}^{[m]}$.

12. If $m < \max(m_{\text{stop},1}, \ldots, m_{\text{stop},K})$, go to step 4. If not, return $\hat{\boldsymbol{\eta}}^{[m]}$.

and $M_{k,\max} = 10$ for all k, and divide each grid into 10 points, i.e., $N_1 = N_2 = N_3 = 10$. The total search grid will consist of $N_1 \cdot N_2 \cdot N_3 = 10^3 = 10000$ tuples of configurations of $\boldsymbol{M}$, enumerated below:

$$\begin{aligned}
&(M_{1,1}, M_{2,1}, M_{3,1}) \\
&(M_{1,1}, M_{2,1}, M_{3,2}) \\
&\dots \\
&(M_{1,1}, M_{2,1}, M_{3,10}) \\
&(M_{1,1}, M_{2,2}, M_{3,1}) \\
&\dots \\
&(M_{1,1}, M_{2,2}, M_{3,10}) \\
&\dots \\
&(M_{1,10}, M_{2,10}, M_{3,10})\,.
\end{aligned}$$

We want to find the best configuration $\boldsymbol{M}$ of all such tuples, i.e., we want to find the minimum of the hyperplane of cross-validated errors $CV(\boldsymbol{M})$. Like in subsection 3.6.1, we must calculate the estimate of the cross-validated prediction error for each given configuration $\boldsymbol{m}$, obtaining the prediction error $\mathrm{CV}(\boldsymbol{m})$. We choose $\boldsymbol{m}_{\text{stop}}$ to be the minimizer of this error,

$$\boldsymbol{m}_{\text{stop}} = \underset{\boldsymbol{m}}{\operatorname{argmin}}\, \mathrm{CV}(\boldsymbol{m}).$$

3.9 Noncyclical component-wise multidimensional boosting algorithm

In the cyclical algorithm seen previously in Algorithm 4, the different $m_{\text{stop},j}$ parameters are not independent of each other, and hence they have to be jointly optimized. As we saw in the previous subsection (3.8.4), the usually applied *grid search* for such parameters scales exponentially with the number of parameters K. This can quickly become very computationally demanding. Thomas et al. (2018) developed a new algorithm for fitting GAMLSS models, in which only one scalar tuning parameter m_{stop} is needed because only one parameter is chosen in each boosting iteration. Thomas et al. (2018) call their new algorithm "noncyclical." Compared to the cyclical algorithm in *gamboostLSS* (Mayr et al., 2012a), this noncyclical algorithm obtains faster variable tuning and equal prediction results on simulation studies carried out (Thomas et al., 2018). We will now explain the necessary adjustments that this algorithm makes.

3.9.1 Gradients are not comparable across parameters

In the cyclical algorithm, we always boost all parameters in the same iteration. Therefore we do not need to choose between parameters. However, if we want to choose one parameter in each boosting iteration, we need to be able to find out which of the parameters would lead to the best increase in performance in this iteration. We are already doing this for choosing which component-wise learner to use for each parameter θ_k. We choose the base learner with the best residual-sum-of-squares (RSS), with respect to the negative gradient vector $u_k^{[m-1]}$. We called this the inner loss. We cannot naïvely extend this criterion to

compare between two parameters, say, θ_1 and θ_2, because the parameters have different scales (Thomas et al., 2018), and therefore (in general) the scale of their respective generalized residual vectors will not be comparable. It is simply not the case that the parameter with the least RSS causes the best decrease in loss. We therefore need a different approach to compare between parameters. In the noncyclical algorithm, we still choose the component-wise base learner which best fits according to the RSS, for each parameter θ_k,

$$\hat{h}_{k,j}(\cdot).$$

After having done so, we calculate the potential improvement in the loss function, which we denote $\Delta\rho_k$,

$$\Delta\rho_k = R\left(\hat{\boldsymbol{\eta}}^{[m-1]} + \nu \cdot \hat{h}_{k,j_k^{[m]}}\right),$$

where $\hat{\boldsymbol{\eta}}^{[m-1]}$ is the current vector of additive predictors. After calculating the improvement $\Delta\rho_k$ of each parameter $k = 1, 2, \ldots, K$, we find out which parameter leads to the best increase. We call this $k^{[m]}$ and it is

$$k^{[m]} = \underset{k \in \{1,2,\ldots,K\}}{\operatorname{argmin}} \Delta\rho_k.$$

We incorporate only the best-fitting learner corresponding for that parameter into the full boosting model, so the model after iteration m is

$$\hat{\boldsymbol{\eta}}^{[m]} \leftarrow \hat{\boldsymbol{\eta}}^{[m-1]} + \nu \cdot \hat{h}^{[m]}_{k,j_k^{[m]}}(\cdot).$$

For all $k \in \{k\colon k \neq k^{[m]}, k = 1, 2, \ldots, K\}$ we do not update their parameter θ_k,

$$\hat{\eta}_k^{[m]} \leftarrow \hat{\eta}_k^{[m-1]}.$$

3.9.2 Criterion for selecting component-wise learner

In the previous subsection, we used RSS, or what we called the inner loss. It makes sense to use this because we choose the component with the best explanation of the error terms. It is, however, not the same criterion that is used to choose between parameters, which might be problematic. Thomas et al. (2018) therefore propose using the loss function ρ directly to choose between component-wise learners as well. They call this the "outer loss." In this case, when choosing a component-wise learner for parameter k, we choose the learner that minimizes the outer loss function, i.e.,

$$j^{[m]} = \underset{j}{\operatorname{argmin}}\, \overline{\text{err}}\left(\hat{\boldsymbol{\eta}}^{[m-1]} + \nu \cdot \hat{h}^{[m]}_{k,j}\right)$$

The individual component-wise learners are still estimated by their usual method, i.e., calculating the negative gradient of the generalized residuals and using the base learner to estimate the models. A schematic overview of the algorithm is given in algorithm 5.

3.9.3 Advantages with noncyclical

For the noncyclical algorithm, the fact that the optimal number of boosting steps, m_{stop}, is always a scalar value, is a major advantage. Finding this tuning parameter can be done fairly quickly with standard cross validation schemes, and most importantly, it scales with with the number of parameters. We therefore choose to use the noncyclical version to construct a gradient boosting algorithm for the FHT model discussed in chapter 2.

algo:multi-noncyclical

Algorithm 5 Multidimensional noncyclical component-wise gradient boosting

1. Start with a data set $D = \{\boldsymbol{x}_i, y_i\}_{i=1}^N$ and a chosen loss function $\rho(y, \boldsymbol{\eta}(\boldsymbol{x}))$, for which we wish to minimize the empirical risk, i.e., the loss function evaluated on the samples,

$$\hat{\boldsymbol{\eta}} = \underset{\boldsymbol{\eta}}{\operatorname{argmin}}\, \overline{\operatorname{err}}(\boldsymbol{\eta}) = \underset{\boldsymbol{\eta}}{\operatorname{argmin}} \sum_{i=1}^{n} \rho(y_i, \boldsymbol{\eta}(\boldsymbol{x}_i)).$$

2. Initialize iteration counter m to 0. Initialize additive predictors to constants $\boldsymbol{\beta}_0 = (\beta_{1,0}, \beta_{2,0} \ldots, \beta_{k,0})$, e.g. to those that jointly maximize the training error,

$$\boldsymbol{\beta}_0 = \underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, \overline{\operatorname{err}}\left(\boldsymbol{\beta}\right).$$

3. Specify a set of base learners $\mathcal{H}_k$ for each dimension $k = 1, \ldots, K$. Specify a step length ν.

noncyclic-step-first-loop

4. Increase m by 1 and set k to 0.

5. Increase k by 1.

6. Compute the negative partial derivative $-\frac{\partial \rho}{\partial \eta_k}$ and evaluate at $\hat{\boldsymbol{\eta}}^{[m-1]}(x_i), i = 1, \ldots, N$, yielding negative gradient vector

$$\boldsymbol{u}_k^{[m-1]} = \left(-\frac{\partial}{\partial \eta_k} \rho\left(y_i, \hat{\boldsymbol{\eta}}^{[m-1]}(\boldsymbol{x}_i)\right) \right)_{i=1}^{N}$$

7. Fit the negative gradient vector $\boldsymbol{u}_k^{[m-1]}$ to each of the p_k components of $\boldsymbol{X}$ separately, using each component's respective base learner. This yields p_k vectors of predicted values, $\left\{\hat{h}_{k,j}^{[m]}\right\}_{j=1}^{p_k}$.

8. Select the best fitting base learner for k, $\hat{h}_{k,j^{[m]}}^{[m]}$, either by

   - the inner loss, i.e., the RSS of the base-learner fit w.r.t the negative gradient vector

$$j^{[m]} = \underset{j \in 1, \ldots, p_k}{\operatorname{argmin}} \sum_{i=1}^{N} \left(u_{k,i} - \hat{h}_{k,j}(\boldsymbol{x}_i)\right)^2$$

   - the outer loss, i.e., the loss function after the potential update,

$$j^{[m]} = \underset{j \in 1, \ldots, p_k}{\operatorname{argmin}}\, \overline{\operatorname{err}}\left(\hat{\boldsymbol{\eta}}^{[m-1]} + \nu \cdot \hat{h}_{k,j}\right)$$

9. Compute the possible improvement of this update regarding the outer loss,

$$\Delta\rho_k = \overline{\operatorname{err}}\left(\hat{\boldsymbol{\eta}}^{[m-1]} + \nu \cdot \hat{h}_{k,j^{[m]}}\right)$$

10. Select the k with boosted predictor that improves training error most,

$$k^{[m]} = \underset{k \in 1, \ldots, K}{\operatorname{argmin}}\, \Delta\rho_k.$$

    Update the additive predictor for this parameter,

$$\hat{\eta}_{k^{[m]}}^{[m]}(\cdot) \leftarrow \hat{\eta}_{k^{[m]}}^{[m-1]} + \nu \cdot \hat{h}_{k^{[m]} j^{[m]}}(\cdot),$$

11. If $m < m_{\text{stop}}$, go to step 4. If not, return $\hat{\boldsymbol{\eta}}^{[m_{\text{stop}}]}(\cdot)$.

Chapter 4

FHTBoost: A twodimensional component-wise gradient boosting algorithm for survival data

ch:FHTboost

In this chapter, we propose a component-wise boosting algorithm for fitting the inverse Gaussian first hitting time (FHT) model (section 2.3 and onward) to survival data. We work with survival data sets with N observations,

$$D = (\boldsymbol{x}_i, \boldsymbol{z}_i, t_i, d_i)_{i=1}^{N}. \tag{4.1}$$

{eq:data-fhtboost}

Here t_i is a possibly right-censored event time, d_i is an indicator variable which is 1 if t_i is an actually observed event time, and 0 if it is right-censored. For simplicity, we sometimes denote the response data

$$y_i = (t_i, d_i).$$

Further, $\boldsymbol{x}_i$ and $\boldsymbol{z}_i$ are covariate vectors that will be related to the inverse Gaussian parameters y_0 and μ, respectively. $\boldsymbol{x}_i$ is defined as

$$\boldsymbol{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,p_1}), \tag{4.2}$$

with p_1 number of covariates, and $\boldsymbol{z}_i$ is defined as

$$\boldsymbol{z}_i = (z_{i,1}, z_{i,2}, \ldots, z_{i,p_2}), \tag{4.3}$$

with p_2 number of covariates. The dimensions of the matrices, p_1 and p_2, can be any size, however our motivating setting is a combination of a high-dimensional p_1, typically gene data, and a low-dimensional p_2, i.e. clinical data. There are theoretical arguments for this particular combination of the high and low-dimensional vectors and parameters, which we have discussed earlier in subsection 2.5.4. Before describing the algorithm in detail, we explain the necessary parts and expressions that will be used in the algorithm.

4.1 Structure of the additive predictors

The first-hitting-time model with Wiener processes, as shown, leads to lifetimes that follow a parameterization of the inverse Gaussian. Hence its log likelihood

depends on y_0, μ, with σ set to 1 to avoid overparameterization (see subsection 2.4.3 for reasoning). Given a data set D (4.1), the log-likelihood of the parameters is

$$l(y_0, \mu|D) = \sum_{i=1}^{N} d_i \left(\ln y_0 - \frac{1}{2}\ln\left(2\pi t_i^3\right) - \frac{(\mu t_i + y_0)^2}{2t_i} \right) + (1-d_i)\ln\left(\Phi\left(\frac{\mu t_i + y_0}{\sqrt{t_i}} \right) - \exp(-2y_0\mu)\Phi\left(\frac{\mu t_i - y_0}{\sqrt{t_i}} \right) \right).$$

We denote the $K = 2$ distribution parameters as

$$\boldsymbol{\theta} = (\theta_1, \theta_2)^T = (y_0, \mu).$$

We wish to use a similar setup as the GAMLSS (see subsection 3.8.1) to model these parameters. We will therefore model the parameters in y_0 and μ through additive predictors η_1 and η_2, respectively. We relate the additive predictors to covariates, and further relate the additive predictors to the parameters by their usual link functions, i.e. the log link for y_0 (to ensure it is positive) and the identity link for the drift μ. In FHT regression, the additive predictors are typically *linear* predictors (Lee and Whitmore, 2006), and we also use this structure. We let the additive predictors for an individual i be defined as

$$\eta_{1,i} \coloneqq \ln(y_{0,i}) = \boldsymbol{\beta}^T \boldsymbol{x}_i = \beta_0 + \sum_{j=1}^{p_1} x_{i,j}\beta_j \tag{4.4}$$

{eq:eta1}

and

$$\eta_{2,i} \coloneqq \mu_i = \boldsymbol{\gamma}^T \boldsymbol{z}_i = \gamma_0 + \sum_{j=1}^{p_2} z_{i,j}\gamma_j, \tag{4.5}$$

{eq:eta2}

and we denote the vector containing both as $\boldsymbol{\eta}_i = (\eta_{1,i}, \eta_{2,i})$. We are going to estimate the additive predictors by gradient boosting, and specifically by estimating the β_j and γ_j parameters. In choosing a linear structure for the additive predictors we also choose base learners in the boosting algorithm which allow for this. We first derive the partial derivatives of the log-likelihood, which will be used to calculate the generalized residuals in each boosting iteration.

4.2 Loss function and its partial derivatives

Since we assume we have observations from a probability distribution, we use the negative of the corresponding log likelihood as our loss function (Mayr et al., 2014a). Given the estimated additive predictor $\hat{\boldsymbol{\eta}}_i$ for an observation $D_i = (\boldsymbol{x}_i, \boldsymbol{z}_i, t_i, d_i)$ from data set D (4.1), we calculate the corresponding estimated distribution parameters by transforming the additive predictors via the inverse of their link functions. Thus,

$$\hat{y}_{0,i} = \hat{\theta}_{1,i} = g_1^{-1}(\hat{\eta}_{1,i}) = \exp(\hat{\eta}_{1,i}), \tag{4.6}$$

and

$$\hat{\mu}_i = \hat{\theta}_{2,i} = g_2^{-1}(\hat{\eta}_{2,i}) = \hat{\eta}_{2,i}. \tag{4.7}$$

For an observation y_i in (4.1), and estimated parameters $\hat{\boldsymbol{\theta}}_i = (\hat{y}_{0,i}, \hat{\mu}_i)$ for this observation, we calculate the loss as

$$\rho(y_i, \hat{\boldsymbol{\theta}}_i) = -l(\hat{y}_{0,i}, \hat{\mu}_i | D_i).$$

This means that as a function of the additive predictors, the loss function for an observation i is

$$\rho(y_i, \hat{\boldsymbol{\eta}}_i) = -l(\exp(\hat{\eta}_{1,i}), \hat{\eta}_{2,i}),$$

where l again is the log-likelihood. To use the gradient boosting algorithm, we need to compute the negative gradient of the loss function, i.e., the negative of the derivative. The negative partial derivative of the loss function is just the partial derivative of the log-likelihood function, i.e. the negatives cancel out. The partial derivative for y_0 is

$$-\frac{\partial}{\partial \eta_1}\rho(y_i, \hat{\boldsymbol{\theta}}_i) = \frac{\partial}{\partial \eta_1} l(\exp(\hat{\eta}_{1,i}), \hat{\eta}_{2,i}) = \hat{y}_{0,i}\frac{\partial}{\partial y_0} l(\hat{y}_{0,i}, \hat{\mu}_i), \tag{4.8}$$

{eq:partial-deriv-y0}

where we used the chain rule because $\eta_1 = \exp(y_0)$. The partial derivative for μ needs no chain rule because the link function is linear, and so it is

$$-\frac{\partial}{\partial \eta_2}\rho(y_i, \hat{\boldsymbol{\theta}}_i) = \frac{\partial}{\partial \eta_2} l(\exp(\hat{\eta}_{1,i}), \hat{\eta}_{2,i}) = \frac{\partial}{\partial \mu} l(\hat{y}_{0,i}, \hat{\mu}_i). \tag{4.9}$$

{eq:partial-deriv-mu}

These partial derivatives turn out to be quite complicated, but we report them here for completeness, in (4.10) and (4.11). See Appendix A for their complete derivation.

4.2.1 Partial derivative for y_0

$$\begin{aligned}
&\frac{\partial}{\partial y_0} l\left(\hat{y}_{0,i}, \hat{\mu}_i\right) \\
&\quad = d_i\left(\frac{1}{\hat{y}_{0,i}} - \frac{\hat{y}_{0,i} + \hat{\mu}_i t_i}{\sigma^2 t_i}\right) + (1 - d_i)\cdot \\
&\quad \left[\frac{1}{\sqrt{t_i}}\phi\left(\frac{\hat{\mu}_i t_i + \hat{y}_{0,i}}{\sqrt{t_i}}\right) + 2\hat{\mu}_i \exp(-2\hat{y}_{0,i}\hat{\mu}_i)\Phi\left(\frac{\hat{\mu}_i t_i - \hat{y}_{0,i}}{\sqrt{t_i}}\right)\right. \\
&\quad \left. + \frac{1}{\sqrt{t_i}} \exp\left(-2\hat{y}_{0,i}\hat{\mu}_i\right) \phi\left(\frac{\hat{\mu}_i t_i - \hat{y}_{0,i}}{\sqrt{t_i}}\right)\right]\cdot \\
&\quad \left[\Phi\left(\frac{\hat{\mu}_i t_i + \hat{y}_{0,i}}{\sqrt{t_i}}\right) - \exp\left(-2\hat{y}_{0,i}\hat{\mu}_i\right) \Phi\left(\frac{\hat{\mu}_i t_i - \hat{y}_{0,i}}{\sqrt{t_i}}\right)\right]^{-1}
\end{aligned} \tag{4.10}$$

{eq:full-partial-deriv-y0}

4.2.2 Partial derivative for μ

$$\begin{aligned}
&\frac{\partial}{\partial \mu} l(\hat{y}_{0,i}, \hat{\mu}) \\
&\quad = d_i(-\hat{y}_{0,i} + \hat{\mu} t_i) + (1 - d_i) \cdot \\
&\quad \left[\frac{t_i}{\sqrt{t_i}} \phi\left(\frac{\hat{\mu} t_i + \hat{y}_{0,i}}{\sqrt{t_i}} \right) + 2\hat{y}_{0,i} \exp(-2\hat{y}_{0,i}\hat{\mu}) \Phi\left(\frac{\hat{\mu} t_i - \hat{y}_{0,i}}{\sqrt{t_i}} \right) \right. \\
&\quad \left. - \frac{t_i}{\sqrt{t_i}} \exp(-2\hat{y}_{0,i}\hat{\mu}) \phi\left(\frac{\hat{\mu} t_i - \hat{y}_{0,i}}{\sqrt{t_i}} \right) \right] \cdot \\
&\quad \left[\Phi\left(\frac{\hat{\mu} t_i + \hat{y}_{0,i}}{\sqrt{t_i}} \right) - \exp(-2\hat{y}_{0,i}\hat{\mu} \Phi\left(\frac{\hat{\mu} t_i - \hat{y}_{0,i}}{\sqrt{t_i}} \right) \right]^{-1}
\end{aligned} \tag{4.11}$$

{eq:full-partial-deriv-mu}

4.2.3 Training error

The training error, which we calculate while boosting, as a function of an estimated additive predictor $\hat{\boldsymbol{\eta}}$, is the sum of the loss function for each individual sample, using the estimated parameters,

$$\overline{\text{err}}\,(\boldsymbol{\eta}) = -\sum_{i=1}^{N} l(\hat{y}_{0,i}, \mu_i). \tag{4.12}$$

{eq:training-error-fht}

4.3 Base learners

We wish to achieve the additive structure of (4.4) and (4.5). We are also in a high-dimensional setting, and we therefore use the component-wise approach (see subsection 3.5.2). We therefore choose regular least squares functions of only one covariate, without intercepts. Each base learner is a function of an observation i, but only uses one covariate in each. This is common in gradient boosting algorithms (Mayr et al., 2014a, see). This means that the base learners corresponding to η_1, $\mathcal{H}_1 = \{h_{1,j}\}_{j=1}^{p_1}$ as a function of $\boldsymbol{x}_i$ is

$$\{h_{1,j}(\boldsymbol{x}_i)\}_{j=1}^{p_1} = \{\beta_j x_{i,j}\}_{j=1}^{p_1}.$$

In an iteration m in the boosting algorithm, to estimate a base learner $\hat{h}_{1,j}^{[m]}$, we use ordinary least squares to calculate its parameters $\hat{\beta}_j^{[m]}$. The vector that we wish to fit is $\boldsymbol{u}_1^{[m-1]}$, the vector of generalized residuals, i.e. the negative derivatives of the loss function, given the model from the previous iteration, for parameter y_0. The estimated parameter $\hat{\beta}_j^{[m]}$ will therefore be

$$\hat{\beta}_j^{[m]} = \left(\boldsymbol{x}_j^T \boldsymbol{x}_j\right)^{-1} \boldsymbol{x}_j^T \boldsymbol{u}_1^{[m-1]} = \frac{\sum_{k=1}^{N} u_{1,k}^{[m-1]} x_{k,j}}{\sum_{k=1}^{N} x_{k,j}^2}.$$

The estimated base learner $\hat{h}_{1,j}^{[m]}$, as a function of an observation

$$\boldsymbol{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,j}, \ldots, x_{i,N}) \tag{4.13}$$

is thus

$$\hat{h}_{1,j}^{[m]}(\boldsymbol{x}_i) = \hat{\beta}_j^{[m]} x_{i,j} = \frac{\sum_{k=1}^{N} u_{1,k}^{[m-1]} x_{k,j}}{\sum_{k=1}^{N} x_{k,j}^2} x_{i,j}.$$

Similarly, the base learners for η_2, are $\mathcal{H}_2 = \{h_{2,j}\}_{j=1}^{p_2}$. Each $h_{2,j}(\cdot)$ uses covariate j from the $\boldsymbol{z}_j$ vector of covariates. As a function of one observation $\boldsymbol{z}_i$, it is given by

$$\{h_{2,j}(\boldsymbol{z}_i)\}_{j=1}^{p_2} = \{\gamma_j z_{i,j}\}_{j=1}^{p_2}.$$

To estimate $\hat{\gamma}^{[m]}$, the same method is used as for $\hat{\beta}^{[m]}$, namely to calculate the ordinary least squares parameter to a vector of generalized residuals. Now, these are $\boldsymbol{u}_2^{[m]}$, the negative derivatives for parameter μ. An estimated base learner $\hat{h}_{2,j}^{[m]}$ is a function which takes an observation $\boldsymbol{z}_i$, and is

$$\hat{h}_{2,j}^{[m]}(\boldsymbol{z}_i) = \hat{\gamma}_j^{[m]} z_{i,j} = \frac{\sum_{k=1}^{N} u_{2,k}^{[m-1]} z_{k,j}}{\sum_{k=1}^{N} z_{k,j}^2} z_{i,j}.$$

In each boosting iteration, all base learners $\hat{h}_{1,j}^{[m]}$ for each covariate $j = 1, \ldots, p_1$ as well as all base learners $\hat{h}_{2,j}^{[m]}$ for each covariate $j = 1, \ldots, p_2$ are estimated. The best fitting base learner for each $k = 1, 2$ is calculated, and then the one of these two that leads to best decrease in the training error, is added to the boosted model, with a step length ν, which we set to 0.1, as per convention.

4.4 Initialization via maximum likelihood estimate

To ensure proper estimation of η_1 and η_2, we need to compute the intercepts β_0 and γ_0, which capture the general, average effect of the covariates. If analytical relationships or formulas exist, such as taking the average of a Gaussian distributed variable in an ordinary regression setting, their computation would be straightforward. In lieu of such known formulas, a reasonable method to use is to perform numerical maximization of the log-likelihood, treating the log likelihood as a function of

$$\boldsymbol{\eta}_0 = (\beta_0, \gamma_0)$$

only. Hence the intercepts are estimated to be

$$\hat{\boldsymbol{\eta}}_0 = \underset{\boldsymbol{\eta}}{\operatorname{argmin}} \, \overline{\operatorname{err}}\left(\boldsymbol{\eta}\right),$$

where the training error $\overline{\operatorname{err}}(\cdot)$ is given in 4.12. We use the routine called `nlm`, which belongs to the base package of `R` (R Core Team, 2013), to do the numerical maximization. As our base learners are only functions with parametric effects and no intercept, these intercepts remain fixed

4.5 The FHTBoost algorithm with fixed intercepts

In the previous chapter, we discussed two versions, namely the noncyclical and the cyclical. The noncyclical is reported to achieve the same results as the cyclical, while being slightly less stable (Thomas et al., 2018). This is made up for by the fact that it is much easier to use due to its single tuning parameter, instead of one for each parameter dimension. While both versions could have been tried, we choose to only use the noncyclical to limit the scope of the investigation in the coming chapters. A schematic overview of the algorithm is given in Algorithm 6. Two helper algorithms are given in Algorithm 7 and Algorithm 8.

algo:fhtboost

Algorithm 6 FHTBoost with fixed intercept

1. Given a survival data set $D = (\boldsymbol{x}_i, \boldsymbol{z}_i, t_i, d_i)_{i=1}^N$, where for each observation i, $\boldsymbol{x}_i$ and $\boldsymbol{z}_i$ are vectors of covariate information,, t_i is the possibly right-censored survival time, and d_i is the censoring indicator. We wish to estimate the $\hat{\boldsymbol{\eta}}$ that minimizes the training error in m_{stop} number of iterations,
$$\hat{\boldsymbol{\eta}} = \underset{\boldsymbol{\eta}}{\operatorname{argmin}}\, \overline{\text{err}}(\boldsymbol{\eta})$$

2. Set iteration counter m to 0. Specify stopping iteration m_{stop}. Initialize additive predictors $\hat{\boldsymbol{\eta}}_0 = (\beta_0, \gamma_0)$, by finding maximum likelihood constants through numerical maximization,
$$\hat{\boldsymbol{\eta}}_0 = \underset{\boldsymbol{\eta}}{\operatorname{argmin}}\, \overline{\text{err}}\left(\boldsymbol{\eta}\right).$$

algostep:FHT-base-learner

3. Specify linear least squares base learners:
$$\mathcal{H}_1 = \{h_{1,j}\}_{j=1}^{p_1} = \{x_j\beta_j\}_{j=1}^{p_1},$$
$$\mathcal{H}_2 = \{h_{2,j}\}_{j=1}^{p_1} = \{z_j\gamma_j\}_{j=1}^{p_2}$$

algostep:FHT-init

4. Increase m by 1.

5. Use Algorithm 7 to calculate base learners corresponding to parameter y_0. Choose the best estimated base learner $\hat{h}_{1,j_1^{[m]}}$, and calculate the possible improvement of the loss function if this base learner is added to the model.

6. Use Algorithm 8 to calculate base learners corresponding to parameter μ. Choose the best estimated base learner $\hat{h}_{2,j_2^{[m]}}$, and calculate the possible improvement of the loss function if this base learner is added to the model.

algostep:FHT-end

7. Find which parameter to include in the boosted model, i.e. the one with smallest gain in error,
$$k^{[m]} = \underset{k\in\{1,2\}}{\operatorname{argmin}}\, \Delta\rho_k.$$
Include this component in the final model, by updating its additive predictor,
$$\hat{\eta}_{k^{[m]}}^{[m]} \leftarrow \hat{\eta}_{k^{[m]}}^{[m-1]} + \nu \cdot \hat{h}_{k^{[m]},\, j^{[m]}}^{[m]}(x),$$
while for the other component $k \neq k^{[m]}$, set the additive predictor for this iteration to be the same as previous iteration
$$\hat{\eta}_k^{[m]} \leftarrow \hat{\eta}_k^{[m-1]}.$$

8. If $m < m_{\text{stop}}$, go to step 4. If not, return
$$\hat{\boldsymbol{\eta}}(\cdot) = \hat{\boldsymbol{\eta}}^{[m_{\text{stop}}]}(\cdot) = \hat{\boldsymbol{\eta}}_0 + \sum_{m=1}^{[m_{\text{stop}}]} \boldsymbol{\eta}^{[m]}(\cdot).$$

algo:fhtboost-minor1

Algorithm 7 Estimate base learners for y_0

1. Compute the negative partial derivative $-\frac{\partial \rho}{\partial \hat{\eta}_1}$ and evaluate at $\hat{\boldsymbol{\eta}}^{[m-1]}(\boldsymbol{x}_i, \boldsymbol{z}_i), i = 1, \ldots, N$, yielding negative gradient vector

$$\boldsymbol{u}_1^{[m-1]} = \left(-\frac{\partial}{\partial \hat{\eta}_1} \rho(y_i, \hat{\boldsymbol{\eta}}^{[m-1]}(\boldsymbol{x}_i, \boldsymbol{z}_i)) \right)_{i=1}^{N}$$

2. Fit the negative gradient vector to each of the p_1 components of $\boldsymbol{X}$ (i.e. to each base learner) separately, using the base learners specified in step 3, so i.e., fitting

$$\hat{\beta}_j^{[m]} = \left(\boldsymbol{x}_j^T \boldsymbol{x}_j \right)^{-1} \boldsymbol{x}_j^T \boldsymbol{u}_1^{[m-1]} = \frac{\sum_{k=1}^{N} u_{1,k}^{[m-1]} x_{k,j}}{\sum_{k=1}^{N} x_{k,j}^2},$$

   and letting $\hat{h}_{1,j}^{[m]}(\boldsymbol{x}) = \hat{\beta}_j^{[m]} x_j$.

3. Select the best fitting base learner, $\hat{h}_{1, j_1^{[m]}}$, by the inner loss, i.e., the RSS of the base-learner fit w.r.t the negative gradient vector

$$j_1^{[m]} = \underset{j \in 1, \ldots, p_1}{\operatorname{argmin}} \sum_{i=1}^{N} \left(u_{1,i}^{[m-1]} - \hat{h}_{1,j}^{[m]}(\boldsymbol{x}_i) \right)^2.$$

4. Compute the possible improvement of this update regarding the outer loss,

$$\Delta\rho_1 = \overline{\text{err}} \left(\hat{\boldsymbol{\eta}}^{[m-1]} + \nu \cdot \hat{h}_{1, j_1^{[m]}}^{[m]} \right) - \overline{\text{err}} \left(\hat{\boldsymbol{\eta}}^{[m-1]} \right)$$

4.6 Resulting model after boosting

After m_{stop} iterations, we have additive predictors $\hat{\eta}_1^{[m_{\text{stop}}]}$ and $\hat{\eta}_2^{[m_{\text{stop}}]}$. As a sum of the boosted additions, they are

$$\hat{\eta}_1^{[m_{\text{stop}}]}(\boldsymbol{x}) = \hat{\beta}_0 + \sum_{m=1}^{m_{\text{stop}}} \nu \cdot \hat{h}_{j^{[m]}}^{[m]}(\boldsymbol{x})$$

and

$$\hat{\eta}_2^{[m_{\text{stop}}]}(\boldsymbol{z}) = \hat{\gamma}_0 + \sum_{m=1}^{m_{\text{stop}}} \nu \cdot \hat{h}_{j^{[m]}}^{[m]}(\boldsymbol{z}).$$

Importantly, they can also be seen as linear predictors of the form (4.4) and (4.5),

$$\hat{\eta}_1^{[m_{\text{stop}}]}(\boldsymbol{x}) = \hat{\boldsymbol{\beta}}^T \boldsymbol{x} = \hat{\beta}_0 + \sum_{j=1}^{p_1} \hat{\beta}_j x_j$$

algo:fhtboost-minor2

Algorithm 8 Estimate base learners for μ

1. Compute the negative partial derivative $-\frac{\partial \rho}{\partial \hat{\eta}_2}$ and evaluate at $\hat{\boldsymbol{\eta}}^{[m-1]}(\boldsymbol{x}_i, \boldsymbol{z}_i), i = 1, \ldots, N$, yielding negative gradient vector

$$\boldsymbol{u}_1^{[m-1]} = \left(-\frac{\partial}{\partial \hat{\eta}_1} \rho(y_i, \hat{\boldsymbol{\eta}}^{[m-1]}(\boldsymbol{x}_i, \boldsymbol{z}_i)) \right)_{i=1}^{N}$$

2. Fit the negative gradient vector to each of the p_2 components of $\boldsymbol{Z}$ (i.e. to each base learner) separately, using the base learners specified in step 3, so i.e., fitting

$$\hat{\gamma}_j = \left(\boldsymbol{z}_j^T \boldsymbol{z}_j \right)^{-1} \boldsymbol{z}_j^T \boldsymbol{u}_2^{[m-1]} = \frac{\sum_{k=1}^{N} u_{2,k}^{[m-1]} z_{k,j}}{\sum_{k=1}^{N} z_{k,j}^2}.$$

3. Select the best fitting base learner, $\hat{h}_{2,j_2^{[m]}}^{[m]}$, by the inner loss, i.e., the RSS of the base-learner fit w.r.t the negative gradient vector

$$j_2^{[m]} = \underset{j \in 1, \ldots, p_2}{\operatorname{argmin}} \sum_{i=1}^{N} \left(u_{2,i}^{[m-1]} - \hat{h}_{2,j}(\boldsymbol{z}_i) \right)^2.$$

4. Compute the possible improvement of this update regarding the outer loss,

$$\Delta \rho_2 = \overline{\text{err}} \left(\hat{\boldsymbol{\eta}}^{[m-1]} + \nu \cdot \hat{h}_{2, j_2^{[m]}}^{[m]} \right) - \overline{\text{err}} \left(\hat{\boldsymbol{\eta}}^{[m-1]} \right)$$

---

and

$$\hat{\eta}_2^{[m_{\text{stop}}]}(\boldsymbol{z}) = \hat{\boldsymbol{\gamma}}^T \boldsymbol{z} = \hat{\gamma}_0 + \sum_{j=1}^{p_2} \hat{\gamma}_j z_j.$$

Here the parameters $\hat{\beta}_0$ and $\hat{\gamma}_0$ are the intercepts calculated initially. Further, the parameters for covariates, $\hat{\beta}_j$ and $\hat{\gamma}_j$, are sums of the boosted parameters. Since each estimated base learner for each parameter j is just a linear effect, we can sum the estimated parameters. In each iteration, we only add the estimated parameter for covariate j if it is the best covariate, i.e. $j_k^{[m]}$ is this j, and we add it with shrinkage ν. Therefore the final parameter $\hat{\beta}_j$

$$\hat{\beta}_j = \sum_{m=1}^{[m_{\text{stop}}]} \nu \cdot I \left(j_1^{[m]} = j \right) \hat{\beta}_j^{[m]},$$

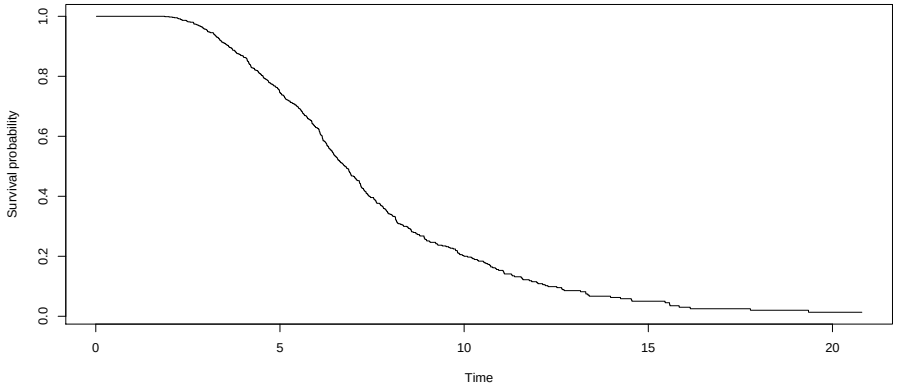
for $j = 1, \ldots, p_1$, and and the final parameter $\hat{\gamma}_j$ is

$$\hat{\gamma}_j = \sum_{m=1}^{[m_{\text{stop}}]} \nu \cdot I \left(j_2^{[m]} = j \right) \hat{\gamma}_j^{[m]},$$

for $j = 1, \ldots, p_2$.

Figure 4.1: Kaplan-Meier plot of generated survival data from subsection 4.7.

fig:small-example-kaplan-meier



4.7 Small example

subsec:algo-example

Let us now consider an example where we draw $N = 500$ survival times from an inverse Gaussian FHT distribution (2.20), and confirm that FHTBoost recovers the maximum likelihood parameters. Let parameter vectors be

$$\boldsymbol{\beta} = (2, 0.1, 0.2)$$

and

$$\boldsymbol{\gamma} = (-1, -0.1, 0.1).$$

We let the covariate matrices be $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2)$ and $\boldsymbol{Z} = (\boldsymbol{z}_1, \boldsymbol{z}_2)$, where all elements in the covariate vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{z}_1, \boldsymbol{z}_2$ are independently drawn from $N(0, 1)$, i.e. completely uncorrelated. We simulate data using Algorithm 10 in section 6.2, with censoring times w_i being drawn from a distribution exp(0.1). We thus end up with a survival data set of the by now usual form

$$D = (\boldsymbol{x}_i, \boldsymbol{z}_i, t_i, d_i)_{i=1}^N.$$

The resulting survival times have Kaplan-Meier estimates shown as the solid black line in Figure 4.1. We estimate $\hat{\boldsymbol{\beta}}$ and $\hat{\boldsymbol{\gamma}}$ using FHTBoost, which we in this case run until convergence. Figure 4.2 shows a plot of the negative log likelihood of the data (training error) as a function of the iteration number m. The solid black line shows negative log-likelihood for FHTBoost with a fixed intercept, i.e. Algorithm 6. In this version with fixed intercept, we only estimate the intercepts $\hat{\boldsymbol{\eta}}_0 = \left(\hat{\beta}_0, \hat{\gamma}_0\right)$ before we begin iterating, and so these intercepts are not changed in any of the boosting iterations. The horizontal solid red line shows the negative of the maximum likelihood, obtained through numerical maximization of the joint maximum likelihood. It is clear that FHTBoost with a fixed intercept does not reach the maximum likelihood value. The maximum likelihood estimated intercepts are $\hat{\beta}_{1,0}^{\mathrm{ML}} = 1.98$ and $\hat{\beta}_{2,0}^{\mathrm{ML}} = -1.02$. FHTBoost with a fixed intercept produces estimates of the intercepts as $\hat{\beta}_0 = 1.68$ and $\hat{\gamma}_0^{[m_{\mathrm{stop}}]} = -0.71$. These estimated intercepts are quite off from the maximum likelihood estimates (see also Table 4.1).

Figure 4.2: Negative log-likelihood for the boosting algorithm with both fixed and changing intercept, as a function of iteration number m

fig:boosting-ML

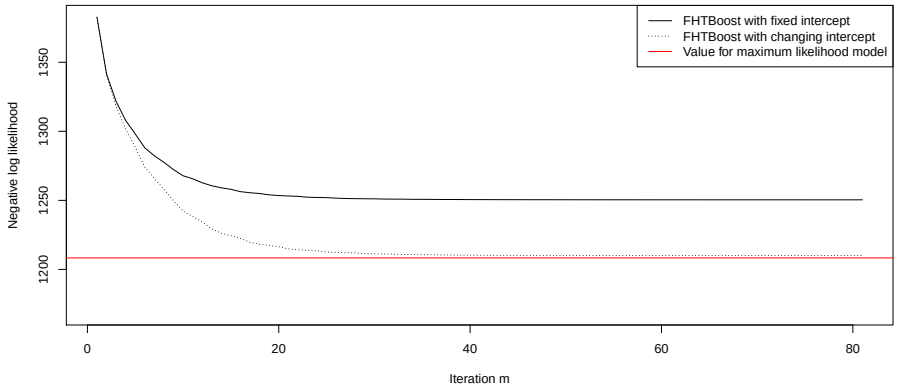


Table 4.1: Parameter values for the example in subsection 4.7

table:ML

	$\hat{\beta}_0$	$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{\gamma}_0$	$\hat{\gamma}_1$	$\hat{\gamma}_2$
Maximum likelihood estimate	1.98	0.10	0.21	-1.03	-0.09	0.12
FHTBoost with fixed intercept	1.68	0.10	0.18	-0.72	-0.06	0.09
FHTBoost with changing intercept	1.97	0.10	0.18	-1.02	-0.09	0.12

To achieve the maximum likelihood value, we found out that we have to modify the algorithm to change intercepts while boosting. Some existing boosting algorithms modify the intercept, but we fount it unclear in the literature how these perform it. *mboost* uses linear least squares base learners with an intercept (Hothorn et al., 2018). One way to change the intercept in each boosting iteration is to treat is as a nuisance parameter, i.e. perform numerical maximization after updating the additive predictor. This will be the same as is done initially to find the intercepts, except now we are using the estimated additive predictors as offsets. Since we will change intercepts in each iteration, we now denote intercepts with their iteration number as well. We write $\hat{\beta}_0^{[m]}$ and $\hat{\gamma}_0^{[m]}$. The initial intercepts are then $\hat{\beta}_0^{[0]}$ and $\hat{\beta}_0^{[0]}$. We replace these in each iteration, with $\hat{\beta}_0^{[m]}$ and $\hat{\beta}_0^{[m]}$, which are calculated in the intercept estimated in step m of the boosting algorithm. By incorporating *changing* intercepts, the algorithm was successful in recovering the maximum likelihood value and parameters. See table 4.1 for the estimated parameter values, and graphically, in figure 4.2, where the convergence is plotted as a function of the number of boosting iterations. When we update the intercept in each iteration the final model has the interpretation that the resulting intercepts are the maximum likelihood intercepts *including* the covariate effects estimated while boosting. Meanwhile, for the fixed intercept, the estimated intercepts are the maximum likelihood intercepts *without* estimated covariate effects.

4.8 FHTBoost algorithm with changing intercept

subsec:FHT-intercept

We modify algorithm 6 with the proposed modification with changing the intercepts in each iteration. We replace step 7 in the fixed intercept algorithm, where we now incorporate changing intercepts for the additive predictors in each boosting iteration. See Algorithm 9 for a schematic overview of this change. Note that while this version of the algorithm did lead to reaching the maximum likelihood value on the simulated example, it is not necessarily better to use this than the fixed intercept version. Since we treat the intercept as a nuisance parameter, there is a risk of overfitting by changing the intercept too much. After all, we are interested in the additive predictors which minimize the test error, not the training error. In fact, as we will see in the simulation study in a later chapter, the fixed intercept version performs worse than the changing intercept version. In the next chapter, we will introduce and explain evaluation measures which we will use to evaluate the performance of FHTBoost.

algo:fhtboost-with-intercept

Algorithm 9 FHTBoost with fixed intercept

7. Find which parameter should be included in the model,

$$k^{[m]} = \underset{k \in \{1,2\}}{\operatorname{argmin}} \Delta\rho_k.$$

Include this component in the final model, by updating its additive predictor,

$$\hat{\eta}^{[m]}_{k^{[m]}} = \hat{\eta}^{[m-1]}_{k^{[m]}} + \nu \cdot \hat{h}^{[m]}_{k^{[m]},\, j^{[m]}}(x),$$

while for the other component $k \neq k^{[m]}$, set the additive predictor for this iteration to be the same as previous iteration

$$\hat{\eta}^{[m]}_{k} = \hat{\eta}^{[m-1]}_{k}.$$

Now, update the intercepts $\hat{\beta}_0$ and $\hat{\gamma}_0$ in the additive predictors by maximizing the training error with the other parts of the estimated additive predictors as offsets. The new offsets are thus

$$\left(\beta_0^{[m]}, \gamma_0^{[m]}\right) = \underset{(\beta_0,\gamma_0)}{\operatorname{argmin}} \overline{\operatorname{err}}\left(\left(\beta_0, \beta_1^{[m]}, \ldots, \beta_{p_1}^{[m]}\right), \left(\gamma_0, \gamma_1^{[m]}, \ldots, \gamma_{p_2}^{[m]}\right)\right)$$

Here each $\beta_j^{[m]}$ is the sum of the shrunk estimated parameters so far, i.e.,

$$\hat{\beta}_j^{[m]} = \sum_{a=1}^{m} \nu \cdot I\left(j_1^{[a]} = j\right) \hat{\beta}_j^{[a]},$$

and similarly for $\gamma_j^{[m]}$.

Chapter 5

Evaluation measures

In this chapter, we will explain different evaluation measures which we will use to assess different aspects of models, later in the thesis.

5.1 Assessing model fit with difference in deviance between an estimated model and a null model

sec:deviance

To comparing the performance of two models of the same kind, e.g., two different estimated FHT models, we can calculate their difference in deviance.

5.1.1 Deviance

In general, the deviance of an FHT model with covariate vectors $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$, which we concatenate as

$$\boldsymbol{\theta} = (\boldsymbol{\beta}, \boldsymbol{\gamma}),$$

is

$$\text{dev}(\boldsymbol{\theta}) = 2 \cdot l(\boldsymbol{\theta}),$$

where $l(\boldsymbol{\theta})$ is the log-likelihood value attained by an FHT model with covariate vector $\boldsymbol{\theta}$. As we know, the better the model fit of $\boldsymbol{\theta}$, the higher the value of $l(\boldsymbol{\theta})$. Deviance is used in the generalized linear models (GLM) framework, where there exist general results related to the deviance. These results, which we will not go into here, are the reason why there is a factor of 2 in the deviance. This factor does not matter for us, but we will use it to adhere to convention.

5.1.2 Difference in deviance

Consider two FHT models $\hat{\boldsymbol{\theta}}_1$ and $\hat{\boldsymbol{\theta}}_2$, where $\hat{\boldsymbol{\theta}}_1$ is a less complex model, i.e., it has fewer, or possibly smaller, covariates than $\hat{\boldsymbol{\theta}}_2$. The difference in deviance between $\hat{\boldsymbol{\theta}}_1$ and $\hat{\boldsymbol{\theta}}_2$, which we denote d, is simply the deviance of the more complex model, $\hat{\boldsymbol{\theta}}_2$, subtracted from the deviance of the less complex model, $\hat{\boldsymbol{\theta}}_1$,

$$d = \text{dev}(\hat{\boldsymbol{\theta}}_1) - \text{dev}(\hat{\boldsymbol{\theta}}_2) = 2 \cdot l(\hat{\boldsymbol{\theta}}_1) - 2 \cdot l(\hat{\boldsymbol{\theta}}_2) = 2\left(l(\hat{\boldsymbol{\theta}}_1) - l(\hat{\boldsymbol{\theta}}_2)\right).$$

Since a more complex model should achieve a higher log-likelihood, we expect the difference of deviance to be negative if the model is a good fit to the data. For our purposes, the difference of deviance is a measure of how much the estimated model improves on a model with no covariates, a so-called null model.

5.1.3 Null model

A model which incorporates no covariate information is called a *null model*, and it will behave the same, no matter the covariate information it is given. A null model thus acts as a baseline to which we can compare our estimated model. If an estimated model does not explain the variation of the data better than the null model, and consequently the chosen model is not a good model fit.

In our case, when using FHTBoost, the null model is an FHT model which only contains the intercepts in the covariate vectors. In our procedure, before we start the boosting iterations, we find the maximum likelihood intercepts of the covariate vectors, β_0 and γ_0. These are found by numerically maximizing the likelihood of the training set. This is the model we would get if we performed FHTBoost with 0 iterations. The covariate vectors of the null model, i.e., the covariate vectors before starting boosting, are thus vectors containing only the intercepts, and all other elements being 0:

$$\hat{\boldsymbol{\beta}}^{[0]}_{\text{train}} = (\overbrace{\hat{\beta}^{[0]}_0, 0, 0, \ldots, 0}^{\text{length } p_1})$$

and

$$\hat{\boldsymbol{\gamma}}^{[0]}_{\text{train}} = (\overbrace{\hat{\gamma}^{[0]}_0, 0, 0, \ldots, 0}^{\text{length } p_2}).$$

For simplicity of notation, we again denote the concatenated vector of these as

$$\hat{\boldsymbol{\theta}}^{[0]}_{\text{train}} = \left(\hat{\boldsymbol{\beta}}^{[0]}_{\text{train}}, \hat{\boldsymbol{\gamma}}^{[0]}_{\text{train}}\right).$$

Similarly, an estimated model, which is estimated by using FHTBoost on the training set with m_{stop} steps, has covariate vectors

$$\hat{\boldsymbol{\beta}}^{[m_{\text{stop}}]}_{\text{train}} = (\hat{\beta}^{[m_{\text{stop}}]}_0, \hat{\beta}^{[m_{\text{stop}}]}_1, \hat{\beta}^{[m_{\text{stop}}]}_2, \ldots, \hat{\beta}^{[m_{\text{stop}}]}_{p_1})$$

and

$$\hat{\boldsymbol{\gamma}}^{[m_{\text{stop}}]}_{\text{train}} = (\hat{\gamma}^{[m_{\text{stop}}]}_0, \hat{\gamma}^{[m_{\text{stop}}]}_1, \hat{\gamma}^{[m_{\text{stop}}]}_2, \ldots, \hat{\gamma}^{[m_{\text{stop}}]}_{p_2}).$$

Again let their concatenation be denoted

$$\hat{\boldsymbol{\theta}}^{[m_{\text{stop}}]}_{\text{train}} = \left(\hat{\boldsymbol{\beta}}^{[m_{\text{stop}}]}_{\text{train}}, \hat{\boldsymbol{\gamma}}^{[m_{\text{stop}}]}_{\text{train}}\right).$$

To avoid overfitting, we calculate the log-likelihood values on data from a separate test set, for reasons we have discussed previously. A model estimated on the training set, $\hat{\boldsymbol{\theta}}_{\text{train}}$, has a likelihood on the test set of

$$l^{\text{test}}\left(\hat{\boldsymbol{\theta}}_{\text{train}}\right) = \sum_{i=1}^{N_{\text{test}}} \rho(y_i, \hat{\boldsymbol{\theta}}). \tag{5.1}$$

Hence the difference in deviance between a fitted model and the null model containing no covariates is

$$d = 2\left(l^{\text{test}}\left(\hat{\boldsymbol{\theta}}^{[0]}_{\text{train}}\right) - l^{\text{test}}\left(\hat{\boldsymbol{\theta}}^{[m_{\text{stop}}]}_{\text{train}}\right)\right).$$

The performance of the estimated model $\hat{\boldsymbol{\theta}}_{\text{train}}^{[m_{\text{stop}}]}$ is good when d is small, meaning "very negative." We may still end up with a *positive* difference of deviance. This means that the null model achieved a higher log-likelihood value than the fully estimated model, which is opposite of what we would expect. This is the typical effect of overfitting, where the model "follows" too much random variability in the training set, and hence performs badly on the test set.

5.2 Variable selection measures

sec:variable-selection

As shown in section 5.2, a component-wise gradient boosting algorithm, like FHTBoost, performs data-driven variable selection. The two major objectives of a component-wise gradient boosting algorithm is to perform selection of variables, and shrinkage of variables. We may wish to measure the first of these two, namely the effectiveness of the variable selection. We will, however, not in this section be able to discuss the shrinkage as well. One way to measure the variable selection is to treat it as a classification problem. In this view, correctly selecting a true variable (adding this variable to the boosting model) is an instance of correct classification. In this section we discuss such classification measures.

5.2.1 Terminology

We denote a variable selected by the algorithm as "positive," or P for short, and a variable that is not selected as "negative," or N for short. Since we know which variables actually affect the response, we know how many of the variables selected are selected correctly, in the sense that they are selected and they have an effect. We call these "true positives," or TP for short. Similarly, we know which variables do not affect the response, and hence we can calculate the number of non-informative variables which were not selected, i.e., true negative, or TN for short. Furthermore, we say that selected variables which in truth do not have an effect, are false positives (FP). Similarly, false negatives (FN) are variables which do have an effect, but which were not selected in the boosting model. We now give explain three such metrics that we will use to determine how well a model performs variable selection.

5.2.2 Classification measures

Sensitivity measures the proportion of the selected variables which are informative. The ideal sensitivity is 1, whereas the worst possible sensitivity is 0.

$$\text{Sensitivity} = \frac{TP}{P} \tag{5.2}$$

{eq:sensitivity}

Specificity measures the proportion of variables *not* selected which were not informative. The ideal specificity is 1, and the worst is 0.

$$\text{Specificity} = \frac{TN}{N} \tag{5.3}$$

{eq:specificity}

False discovery rate measures the proportion of selected variables which are in truth not informative. The ideal false discovery rate is 0, and the worst is 1.

Furthermore, we might note that if the rate is above 0.5, then there are more false positives than true positives in the variable selection.

$$\text{FDR} = \frac{FP}{FP + TP} \tag{5.4}$$

{eq:accuracy}

5.3 Evaluating survival prediction with the Brier score

sec:brier

The Brier score (Brier, 1950) was first introduced as a way to measure the accuracy of weather forecasts, and later translated into survival analysis (Graf et al., 1999). Let us first consider, for ease of presentation, a survival data case where there is

5.3.1 Brier score on uncensored survival data

Consider a test set of N_{test} individuals. For all observations $i = 1, \ldots, N_{\text{test}}$, the test set contains an observed survival time t_i, and a covariate vector $\boldsymbol{x}_i$. The estimated survival probability of individual i at time t^* is

$$\hat{\text{S}}(t^*|\boldsymbol{x}_i), \tag{5.5}$$

{eq:phat}

where the prediction function $\hat{\text{S}}(t^*|\boldsymbol{x}_i)$ is obtained from an FHT model. The Brier score aims to evaluate how well $\hat{\text{S}}(t^*|\boldsymbol{x}_i)$ (5.5) is able to predict the event status of individual i at a given time t^*, namely

$$I(t_i > t^*). \tag{5.6}$$

The error made in predicting the event status for a patient in the test set can be given as

$$\begin{aligned} BS(t^*) &= \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \left(I(t_i > t^*) - \hat{\text{S}}(t^*|\boldsymbol{x}_i) \right)^2 \\ &= \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \left[\hat{\text{S}}(t^*|\boldsymbol{x}_i)^2 I(t_i \leq t^*) + (1 - \hat{\text{S}}(t^*|\boldsymbol{x}_i))(I(t_i > t^*) \right]. \end{aligned}$$

In the first formulation, the Brier score is similar to a residual-sum-of-squares (RSS) measure, since we sum the squared error between the observed event and the estimated probability. In the second formulation, we see that the $BS(t^*)$ measure is the average of the Brier score of events that have occurred before time t^*, and events that have not yet occurred. In the case of censored data, however, the above is not enough.

5.3.2 Brier score on censored survival data

The Brier score was consequently adapted to handle censored survival times by Graf et al. (1999). A crucial assumption is that the censored survival times have the independent censoring property (see subsection 2.1.1). They showed that the loss of information due to censoring can be accounted for by using an

inverse probability of censoring weighting (Graf et al., 1999). This version of the Brier score for censored data is defined as

$$BS^c(t^*) = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{\hat{S}(t^*|\boldsymbol{x}_i)^2 I(t_i \leq t^*, \delta_i = 1)}{\hat{G}(t_i)} + \frac{(1-\hat{S}(t^*|\boldsymbol{x}_i))^2 (I(t_i > t^*)}{\hat{G}(t^*)}\right]. \tag{5.7}$$

{eq:brier}

Here $\hat{G}(\cdot)$, defined as

$$\hat{G}(t) = \prod_{i \in \overline{R}(t)}\left(1 - \frac{1-\delta_i}{\sum_{i=1}^{N} Y_i(t)}\right),$$

is the Kaplan-Meier estimate of the *censoring distribution* (Bøvelstad and Borgan, 2011), and where further $Y_i(t)$ is an indicator of whether individual i is at risk at time t, and where $\overline{R}(t)$ is the set of individuals *not* at risk at time t, i.e.,

$$\overline{R}(t) = \{i \colon t_i < t, i = 1, 2, \ldots, N\}.$$

If we look closely at (5.7), we see that the $BS^c(t^*)$ measure is a weighted average of the Brier score of events having happened at or before time t^*, and the events that have not yet occurred.

5.3.3 Integrated Brier score

subsec:
integrated-brier

The Brier score $BS^c(\cdot)$ (5.7) only considers a given timepoint. We might also be interested in a time interval, say, from time t_{start} to time t_{end}, The integrated Brier score is simply the Brier score integrated over this time interval, but divided by the length of the time interval,

$$\text{IBS}(t_{\text{start}}, t_{\text{end}}) = \frac{1}{t_{\text{end}} - t_{\text{start}}}\int_{t_{\text{start}}}^{t_{\text{end}}} BS^c(t)\ \mathrm{d}t.$$

This can be seen as the average Brier score over this interval. In practice, we can obtain $\text{IBS}(t_{\text{start}}, t_{\text{end}})$ by computing the Brier score 5.7 over a grid of time points $t \in [t_{\text{start}}, t_{\text{start}}]$, and then numerically integrating these. If we have survival times $t_1, \ldots, t_N$, we can calculate the Brier score of each such time point t_i, $i = 1, \ldots, N$, and thus calculate an approximation of $\text{IBS}(t_1, t_N)$, by numerical integration over these time points.

$$\widehat{\text{IBS}}(t_1, t_N) \approx \frac{1}{t_N - t_1}\sum_{k=1}^{N-1} BS^c(t_k) \cdot (t_{k+1} - t_k).$$

This approximation is the same as summing up the area in figures of the same type as Figure 7.4. To get a better approximation of the integrated Brier score, we would need to assess the Brier score for more time points, but we did not do this.

Chapter 6

Simulation study

We wish to verify that FHTBoost (see chapter 4) works in a realistic setting, i.e. that it is able to have predictive power and select correct variables. Not only that, but we wish to see which of the two versions of the algorithm works best, namely the fixed intercept version and the version with an iteratively changing intercept. A common way to verify that a statistical estimation method works, is to test it on simulated data. Simulation studies are used for many different purposes, but a particularly common purpose is to simulate survival data from the "true model." In this case the true model is a first-hitting-time model with a Wiener health process which is dependent on y_0 and μ, where we know the true values of the parameters. We can then use our developed statistical estimation method to estimate these parameters, and see how well it recovers the true parameters in the final model, and how well the model fits the simulated data. The estimated model should fit be able to the simulated data well, because the data generating mechanism is the same as what the model assumes. We now describe the simulation design.

6.1 Simulation design

In this section we describe the two scenarios we will examine a highly correlated scenario and an uncorrelated scenario. The motivating setting for both scenarios is a survival analysis setting where we have a high-dimensional covariate matrix $\boldsymbol{X}$, which typically consists of gene expression data, and we have a low-dimensional covariate matrix $\boldsymbol{Z}$, consisting of clinical measurements. The motivation for this dichotomy is discussed in subsection 2.5.4. $\boldsymbol{X}$ is a matrix of covariate vectors

$$\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{p_1},$$

and similarly, $\boldsymbol{Z}$ is a matrix of covariate vectors

$$\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_{p_2}.$$

We let each matrices correspond to a parameter vector, $\boldsymbol{X}$ to $\boldsymbol{\beta}$, and $\boldsymbol{Z}$ to $\boldsymbol{\gamma}$. We link these to the parameters y_0 and μ, respectively, with the log link function and the identity link function, as usual. In each parameter vector, we only set a small number of parameters to a non-zero value, and we set all the rest to zero. Thus only a very small number of covariates will have

an effect. This allows us to assess the effectiveness of the variable selection that FHTBoost performs on the training sets. Given these parameter vectors and covariate matrices, we calculate a specific $y_{0,i}$ and a specific μ_i for each individual i, where $i = 1, 2, \ldots, N$, and where N is the size of the data set. From the FHT perspective, discussed previously in section 2.3.1 and onward, these are parameters representing the health process of each individual. To reiterate, an individual i has a stochastic health process, specifically a Wiener process with an initial level $y_{0,i}$ and a drift μ_i. When the health process hits 0, it causes the event for the individual. Since the FHT of a Wiener with health level $y_{0,i}$ and drift μ is $\text{IG}(y_{0,i}, \mu_i)$, the individual's lifetime follows the distribution $\text{IG}(y_{0,i}, \mu_i)$. This relationship allows us to easily draw a lifetime for each individual from its respective inverse Gaussian distribution.

In each scenario, we generate $B \approx 500$ separate training sets by drawing survival data according to algorithm 10. To generate the covariate matrices $\boldsymbol{X}$ and $\boldsymbol{Z}$ in each training set we will use the method described in Section 6.3, which is a method for simulating clinical and gene data together. Briefly, it consists of specifying blocks of correlated covariates, such that the matrices are drawn according to this correlation structure. The uncorrelated scenario will not have any such blocks, but the correlated scenario will use the block structure. We specify corresponding true parameter vectors, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$. Then we combine these with the true covariate matrices to get vectors $\boldsymbol{y}_0$ and $\boldsymbol{\mu}$, where the i-th element in each of these corresponds to the health process of individual i, where $i = 1, \ldots, N$. Then we draw from the Inverse Gaussian distribution according to Algorithm 10, obtaining N right-censored lifetimes. Hence we have N tuples

$$(\boldsymbol{x}_i, \boldsymbol{z}_i, t_i, d_i)_{i=1}^N.$$

In each data set there are $N = 500$ observations, drawn with a unique seed for each training set, to ensure reproducibility. Given such a training set, we can use the FHT boosting algorithm to estimate parameters. We quickly explain the procedure used. We first use repeated (5 times) 10-fold cross-validation to find the optimal number of boosting steps, m_{stop}, for this data set. As shown in subsection 3.4.3.2, repeating the cross-validation reduces the variance of the estimator of m_{stop}. Then we estimate the model on the whole of this training set, using m_{stop} iterations. It is important to evaluate an estimated model's performance on a separate and unseen test set. We set $N_{\text{test}} = 1000$ observations. The data in the teset here are drawn in the exact same manner as the training data, again with a specific seed for reproducibility. To assess the model fit on we calculate the difference of deviance on a test set, as explained in 5.1. To assess variable selection, we calculate the metrics explained in 5.2. We first discuss the algorithms we use to generate FHT survival data.

6.2 Simulation of survival data from an IG FHT distribution

sec:simulate-IG-data

We wish to simulate survival times $(t_i)_{i=1}^N$ with censoring. We first draw N uncensored survival times $\{\tilde{t}_i\}_{i=1}^N$ from a survival time distribution $f(\cdot)$. If this distribution has a closed form probability distribution function, we can draw from it directly, and this is the case for us. To implement censoring of the data, we draw censoring times $\{w_i\}_{i=1}^N$ from a different lifetime distribution where,

importantly, the parameters do *not* depend on covariates. Thus the observed survival times are

$$t_i = \min(\tilde{t}_i, w_i),$$

as we have seen before. The corresponding censoring indicator, d_i, is then set equal to 1 if the actual survival time was observed, i.e., if $t_i < w_i$. We end up with a data set

$$D = (\boldsymbol{x}_i,\, \boldsymbol{z}_i,\, t_i,\, d_i)_{i=1}^N.$$

Note that this prcedure simulates censored time under independent censoring, since indeed the censoring times are independent of the survival times. Algorithm 10 gives a schematic overview of this procedure.

algo:FHT-sim

Algorithm 10 Generating survival data from Inverse Gaussian FHT distribution

1. Obtain the design matrices $\boldsymbol{X}$, $\boldsymbol{Z}$ and the true parameter vectors $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$.

algo:FHT-sim-step-cens

2. Specify a censoring time distribution.

3. Calculate the distribution parameters y_0 and μ using the link functions,

$$y_0 = \exp(\boldsymbol{\beta}^T \boldsymbol{X}) = \exp\left(\beta_0 + \sum_{j=1}^{p} \beta_j x_j\right),$$

$$\mu = \boldsymbol{\gamma}^T \boldsymbol{Z} = \gamma_0 + \sum_{j=1}^{d} \gamma_j z_j.$$

4. Draw N uncensored survival times $(\tilde{t}_i)_{i=1}^N$ from $\mathrm{IG}(\boldsymbol{\mu}, \boldsymbol{y}_0)$, where

$$\tilde{t}_i \sim \mathrm{IG}(\mu_i, \boldsymbol{y}_{0,i}).$$

5. Draw N censoring times $(w_i)_{i=1}^N$ from the censoring time distribution specified in step 2.

6. Right censor the survival times by choosing

$$t_i = \min(\tilde{t}_i, w_i).$$

   The censoring indicator on whether observation i was observed or not is then

$$d_i = I(t_i = \tilde{t}_i).$$

7. The simulated data set is $D = (t_i,\, d_i)_{i=1}^N$.

6.3 Generating correlated clinical and gene expression data

sec:generating-correlated-data

In a realistic survival analysis data set, there is a lot of correlation. There may correlation between different genes, between different clinical measurements, and also between genes and clinical measurements. To mimic this, we can set up a correlation structure that lets us generate matrices $\boldsymbol{X}$ and $\boldsymbol{Z}$ which have this structure. One way to do this is to imagine that we have blocks of covariates, where inside one block, the covariates are highly correlated, whereas covariates in one block are not correlated to other genes. We set up such a block structure where we have a certain number of blocks, say, B. Each block contains a gene block and a clinical block, and elements in a block are correlated. In each block $p = 1, \ldots, B$, there are a number of genes, G_p, and a number of clinical measurements, C_p. In the gene part of the block, the genes are correlated by $\rho_{g,p}$. In the clinical part of the block, the genes are correlated by $\rho_{c,p}$. There is also a correlation between the gene part of the block and the clinical part of the block, $\rho_{b,p}$. Given all blocks and their correlations, we are able to construct a covariance matrix, such that we can draw normally distributed data which follows the given correlation structure. For the genes and clinical measurements not belong to any block, their correlation is 0. Hence if no blocks are specified, the data will simply be uncorrelated.

We were so kind as to receive code used previously by Riccardo de Bin, which generates matrices $\boldsymbol{X}$ and $\boldsymbol{Z}$ with the structure explained above. The code for this is given in the Appendix, in B.1. Briefly, the code works such that it sets up the covariance matrix, and then draws the matrices from a normal distribution. Afterwards, noise is added to all elements in $\boldsymbol{X}$, both multiplicative noise and additive noise. Algorithm 11 contains a schematic overview of this.

algo:clinical-sim

Algorithm 11 Generating correlated clinical and gene expression data

1. Given number of blocks, B, and corresponding sizes, i.e. for each block $p = 1, \ldots, B$, given a number of genes in the block, G_p, and a number of clinical measurements in the block, C_p. Further, given correlations between genes in block p, $\rho_{g,p}$, and clinical measurements in block, $\rho_{c,p}$. Finally, also given the correlation between the gene part of the block and the clinical part of the block, $\rho_{b,p}$.
2. Set up a covariance matrix corresponding to above.
3. Generate the data, drawing from a normal distribution. The mean values for each gene is 6, and clinical 1.
4. Add noise to $\boldsymbol{X}$, both multiplicative and additive.
5. Threshold values in $\boldsymbol{X}$.

6.4 Scenarios

6.4.1 Scenario 1: Uncorrelated case

We generate 416 data sets of size $N = 500$, and we let $\boldsymbol{X}$ consist of 10000 covariate columns and $\boldsymbol{Z}$ of 15 columns. Hence $\boldsymbol{\beta}$ is of size $p = 10001$, and $\boldsymbol{\gamma}$ is of size $d = 16$. We first discuss the size of the parameter effects we chose. The parameter vector $\boldsymbol{\beta}$ is linked to the initial level y_0 by an exponential link function. Consequently, each parameter effect is multiplicative instead of additive. A large gene expression value can therefore potentially cause a large change in y_0. Since there are 35 elements of $\boldsymbol{\beta}$ which are informative, there is a reasonable chance of such extreme values occurring in y_0. Because of this, we had trouble setting up simulations in which FHTBoost managed to pick up much signal from the underlying parameter vector. The problem was often that the survival times were small, because of these effects being large. Therefore we choose 0.1 for the informative parameter effects $\beta_j, j = 1, 2, \ldots, 35$. The parameter sizes on the drift might also appear rather small, at 0.1, but this effect is linear with time. By having a relatively low effect per time unit, we ensure that the lifetimes are not too short. These two considerations in mind made us achieve a decent simulation setup. Specifically, we set the intercept term in $\boldsymbol{\beta}$ to 2.0, and the following 35 elements to 0.1. We set all other elements to 0. For $\boldsymbol{\gamma}$, we set the intercept term to be -1. In similar fashion as in $\boldsymbol{\beta}$, we let the first 5 elements in $\boldsymbol{\gamma}$ have a non-zero value of -0.1, while we set the remaining 10 elements to 0. Hence, the true parameter vectors are

$$\boldsymbol{\beta} = \left(2.0, \underbrace{0.1, 0.1, \ldots, 0.1}_{\text{length } 35}, \overbrace{0, 0, \ldots, 0}^{\text{length } 9965}\right) \text{ and}$$

$$\boldsymbol{\gamma} = \left(-1.0, \underbrace{0.1, 0.1, \ldots, 0.1}_{\text{length } 5}, \overbrace{0, 0, \ldots, 0}^{\text{length } 10}\right).$$

We generate X and Z from Algorithm 11 for drawing clinical and gene data, with $B = 0$ blocks. We specify that all correlations are 0, i.e.

$$\rho_{g,p} = \rho_{c,p} = \rho_{b,p} = 0,$$

meaning no covariate correlates with any other. We generate 416 data sets from this algorithm, where we set the seed at the beginning of each simulation.

6.4.2 Scenario 2: Correlated case

In this setup, we will let the part of the data that is informative be highly correlated. Similar to in the uncorrelated scenario, we will still assume that we have a large number of genes, but only a small minority of these are informative. We now generate 364 data sets of size $N = 500$, where the "gene" matrix $\boldsymbol{X}$ has $p_1 = 10000$ covariates, i.e. the same as in scenario 1, and the dimensionality of the "clinical" data is $p_2 = 25$, i.e. a bit more than in scenario 1. We again use the method described in Section 6.3 to generate the matrices. We specify $B = 10$ blocks. In each block $p = 1, \ldots, B$, the correlation between genes is set to $\rho_{g,p} = 0.7$, the correlation between clinical measurements is set to $\rho_{c,p} = 0.7$,

table:block-setup

Table 6.1: Block setup for the correlated scenario

	Gene covariates										Clinical covariates		
Block 1	0.1	0.1	0.1	0	0	0	0	0	0	0	-0.1	0	
Block 2	0.1	0.1	0.1	0.1	0	0	0	0	0	0	-0.1	0	
Block 3	0.1	0.1	0.1	0	0	0	0	0	0	0	-0.1	0	
Block 4	0.1	0.1	0.1	0.1	0	0	0	0	0	0	-0.1	0	
Block 5	0.1	0.1	0.1	0	0	0	0	0	0	0	-0.1	0	0
Block 6	0.1	0.1	0.1	0.1	0	0	0	0	0	0	-0.1	0	0
Block 7	0.1	0.1	0.1	0	0	0	0	0	0	0	-0.1	0	0
Block 8	0.1	0.1	0.1	0.1	0	0	0	0	0	0	-0.1	0	0
Block 9	0.1	0.1	0.1	0	0	0	0	0	0	0	-0.1	0	0
Block 10	0.1	0.1	0.1	0.1	0	0	0	0	0	0	-0.1	0	0

and the correlation between the genes and the clinical measurements is also set to $\rho_{b,p} = 0.7$. Of the gene data, 100 genes are part of a block, while the other 9900 genes are not part of any block. These 9900 genes are not informative, in the sense that the corresponding β coefficient is set to 0. Of these 100 genes, only 35 are informative, but the 65 others are part of the correlation structure. We let all 25 clinical covariates be part of a block, but only 10 of these are informative, with a corresponding γ_j coefficient of -0.1. Each block consists of 10 genes and either two or three clinical measurements. Only one (the first) of the clinical measurements in each block is informative. In the gene blocks, either three or four (the first) of the genes are informative. Each informative gene has a corresponding β_j coefficient size of 0.1. The first five of the clinical blocks consist of an informative parameter and one which is not informative, while the last five have one informative and *two* not informative parameters. For the gene blocks, we alternate between blocks with three informative genes, and four informative genes. Table 6.1 shows this setup visually. Each row in the table corresponds to a block, and each column to one covariate.

For the sizes of the intercepts, we also chose similar sizes as in the uncorrelated scenario. The intercept for y_0 is $\beta_0 = 2$, i.e. an individual with all mean gene values will have an initial value $y_0 = \exp(2) = 7.389$. The intercept for μ is -0.5.

6.5 Results

In each scenario, there are many training sets. After having estimated an FHT model each training set, we assess the performance of the model trained on this training set, on the test set. We calculate the difference of deviance. To assess the variable selection of the model, we calculate metrics for the variables selected based on the training set. We count TP, the number of informative covariates which were selected in the estimated boosting model, and TN, the number of non-informative covariates which were selected in the model. Based on these numbers, we calculate the metrics discussed in Section 5.2, namely Sensitivity, Specificity and False Discovery Rate. Let us now consider the results of each scenario.

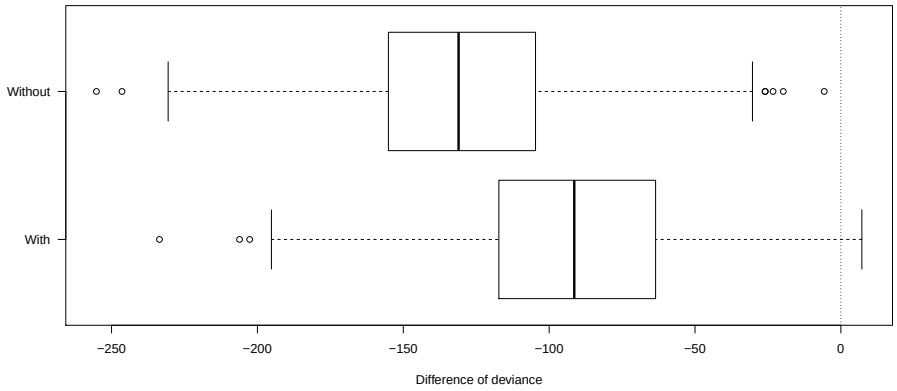
Table 6.2: Difference of deviance results for FHTBoost, uncorrelated case

table: uncorrelated-deviance

	Updating	Fixed
Mean	-92.0	-130.1
Standard deviation	41.8	40.7
Minimum	-233.6	-255.2
Maximum	7.2	-5.7

Figure 6.1: Boxplot for difference in deviance for the two intercept variants, non-correlated scenario

fig:simulation-uncorrelated-deviances-boxplot



6.5.1 Scenario 1: Uncorrelated case

6.5.1.1 Model fit on the test set

The most evident result is that, in contrast to our expectation, the fixed intercept version of FHTBoost seems to perform on average better than that with updating intercept (see Table 6.6). The mean difference of deviance on the updating intercept version is -92.0, while it is -130.1 on the fixed intercept version. Despite the large variability (see min, max and standard deviation in Table 6.6) in the distributions of the difference of deviance results, the difference between the fixed intercept and the changing intercept are noticeable. This is perhaps most apparent in Figure 6.1. In addition, the fixed intercept version always performs better than the null model, while there are a few cases in which the updating intercept version does not (Table 6.6, row "min", and Figure 6.1). In Figure 6.1, we observe that for the fixed intercept, all points are to the left of the dashed vertical line, indicating a deviance of 0). In other words, all models estimated using the fixed intercept version performed better than their null models, whereas this is not the case for the version with a changing intercept. This can be a consequence of overfitting. Moving the intercept allows the model to fit the training data too well. We will now consider the variable selection metrics.

Table 6.3: High dimensional (genomic) part: Performance of FHTBoost in terms of variable selection, uncorrelated case

table: uncorrelated-y0

	Updating		Fixed	
	Mean	Standard error	Mean	Standard error
Sensitivity	0.190	(0.090)	0.452	(0.162)
Specificity	1.000	(0.000)	0.997	(0.002)
FDR	0.310	(0.176)	0.613	(0.144)

Table 6.4: Low dimensional (clinical) part: Performance of FHTBoost in terms of variable selection, uncorrelated case

table: uncorrelated-mu

	Updating		Fixed	
	Mean	Standard error	Mean	Standard error
Sensitivity	0.741	(0.232)	0.958	(0.099)
Specificity	0.943	(0.110)	0.638	(0.291)
FDR	0.091	(0.144)	0.375	(0.192)

Table 6.5: Optimal iteration number m_{stop} results for FHTBoost, uncorrelated case

table: uncorrelated-mstop

	Updating	Fixed
Mean	15.8	63.8
Standard deviation	6.4	26.5
Minimum	2	2
Maximum	39	160

6.5.1.2 Variable selection

Let us contrast the two versions of FHTBoost in terms of variable selection as well. We report the results for the high-dimensional part in Table 6.3, and the low-dimensional part in Table 6.4. This scenario has 35 informative gene covariates and 5 informative clinical covariates. We are considering the sensitivity and specificity on both covariate vectors at the same time. Since we are only selecting a covariate for either $\boldsymbol{\beta}$ or $\boldsymbol{\gamma}$ in each iteration of the boosting algorithm, these scores will depend on each other. Furthermore, since the changing intercept version has a rather low number of iterations, with a mean of 15.8, it is usually impossible to get anywhere near perfect on these metrics, as at most one new parameter is selected in each iteration, and we have 40 informative covariates in total from the two vectors. Since the covariate vector is estimated on the training data, a likely explanation is that the changing intercept captures more of the variation in the training data. In doing so, there is less variation to be explained by the covariates, and hence the boosting algorithm will start to overfit more quickly.

Consider first the result of the version in which the intercept is changed in each step. A very high specificity is achieved for both covariate vectors. The specificity measures the amount of true negatives which are correctly classified as negatives, i.e., not selected. The changing intercept version achieves a mean specificity of 1.00, and a mean false discovery rate of 0.316. Thus there is about

a one in three chance that it selects a variable which is not actually informative. The mean sensitivity, i.e., the ratio of correctly selected informative variables, is only 0.190. This means that a large proportion of the informative covariates are not selected. For $\boldsymbol{\gamma}$, which is related to the clinical covariates, a much higher specificity is attained, with a mean of 0.943. Even though the parameter effects on the drift are rather small, the informative covariates are often correctly selected. Furthermore, the false discovery rate here is very low, with a mean of 0.091.

Now consider the results of the fixed intercept version. For the $\boldsymbol{\beta}$ covariate vector, a higher proportion of informative covariates are selected than the other version of FHTBoost, with a mean sensitivity of 0.452. The mean specificity is 0.997, which is good. Simultaneously, a larger mean false discovery rate (0.613) than the changing intercept version is not good: An FDR of 0.613 means that more than half of all selected variables should be expected to be false positives. At the same time, a very good sensitivity is achieved on the $\boldsymbol{\gamma}$ covariate vector, with a mean of 0.958, and a good specificity with a mean of 0.638. The false discovery rate on $\boldsymbol{\gamma}$ is slightly above one in three, with a mean of 0.375.

6.5.1.3 Summary

Both FHTBoost versions increase the model fit when including covariates, in almost all cases. Both also suffer from a high false discovery rate. This issue has been raised many times before (Bühlmann and Hothorn, 2007; Bühlmann and Yu, 2003; Thomas et al., 2018). It may especially be a problem in a setting such as ours where we have two parameters which are dependent on covariates, as it may lead to overfitting. It also seems that the changing intercept version overfits more than the fixed intercept version, since it is allowed to change the intercept. One remedy may be to use base learners which incorporate intercepts, as this allows also the additional intercepts to be added with the step length ν.

6.5.2 Scenario 2: Correlated case

We now consider the results for the correlated simulation study.

6.5.2.1 Model fit on the test set

We observe in Figure 6.2, which is a boxplot of the deviances in this scenario, that the mean deviance is slightly better for the fixed intercept version, and with better extreme values, both minimum and maximum. The mean deviance for the fixed intercept version is -58.8, while it is -57.8 for the changing intercept version, and both have high variability. We should therefore not say that there is any difference in their performance with regard to model fit in this scenario. One explanation for the equal performance in this scenario might be that since both boosting algorithm stop rather quickly (i.e. small m_{stop}) in this scenario, the intercept is allowed to change less than in scenario 1, and thus there is less overfitting.

6.5.2.2 Variable selection

We now consider the variable selection metrics. We first look at the covariate vector $\boldsymbol{\beta}$ (see Table 6.7), which affects the initial level y_0, and which is related to

Figure 6.2: Boxplot for difference in deviance for the two intercept variants in the correlated scenario. The boxes are without and with changing the intercept.

fig:simulation-correlated-deviances-boxplot

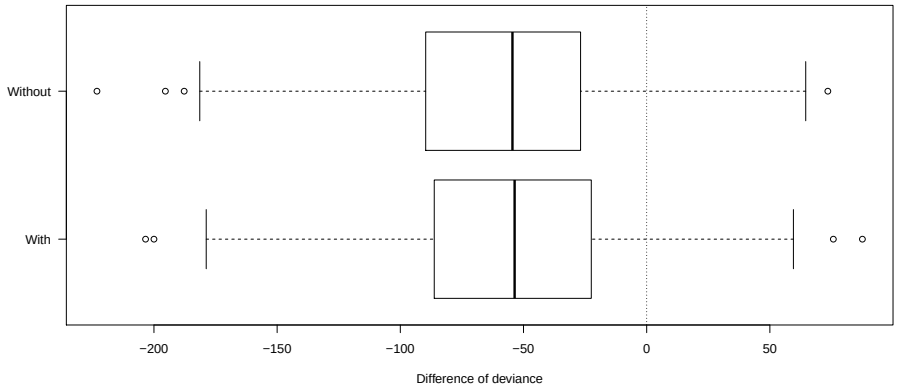


Table 6.6: Difference of deviance results for FHTBoost, correlated case

table: uncorrelated-deviance

	Updating	Fixed
Mean	-57.8	-58.8
Standard deviation	47.2	46.1
Minimum	-203.4	-223.1
Maximum	87.6	73.5

"genomic" variables. The fixed intercept version is slightly better with regards to sensitivity, with a mean of 0.204 versus a mean of 0.157 for the changing intercept version. So selecting informative variables, This does come at the cost of a higher false discovery rate, with a mean as high as 0.652. Here the changing intercept version has a mean of 0.439. The specificity score is almost perfect in both cases.

We now consider the "clinical" covariates, used in $\boldsymbol{\gamma}$ and related to the drift μ. The results for it are reported in Table 6.8. The changing intercept version performs quite a lot worse here than the fixed intercept version, overall. It achieves a mean sensitivity of 0.273, while the fixed intercept version achieves 0.625. However, the changing intercept then achieves a mean of 0.831 for specificity, where the fixed intercept version attains 0.537. For the false discovery rate, the fixed intercept has a slightly higher mean, at 0.507, whereas the changing intercept version has a mean of 0.454.

For this scenario, as well, we observe that the fixed intercept version has a higher optimal iteration number. The mean m_{stop} is 50 with a fixed intercept, compared to a mean of 19.5 for the changing intercept version. Again this necessarily means that more variables are selected, and that coefficients from the changing intercept version will be more shrunken.

Table 6.7: High dimensional (genomic) part: Performance of FHTBoost in terms of variable selection, correlated case

table: correlated-y0

	Updating		Fixed	
	Mean	Standard error	Mean	Standard error
Sensitivity	0.157	(0.084)	0.204	(0.081)
Specificity	0.999	(0.000)	0.998	(0.001)
FDR	0.439	(0.218)	0.652	(0.181)

Table 6.8: Low dimensional (clinical) part: Performance of FHTBoost in terms of variable selection, correlated case

table: correlated-mu

	Updating		Fixed	
	Mean	Standard error	Mean	Standard error
Sensitivity	0.273	(0.187)	0.625	(0.245)
Specificity	0.831	(0.139)	0.537	(0.236)
FDR	0.454	(0.250)	0.507	(0.130)

Table 6.9: Optimal iteration number m_{stop} results for FHTBoost, correlated case

table: correlated-mstop

	Updating	Fixed
Mean	20.0	51.1
Standard deviation	12.1	24.4
Minimum	2	2
Maximum	65	148

6.5.2.3 Summary

We would expect that there would in general be higher false discovery rates in this scenario, since the data are correlated, and thus covariates which are not really informative actually often carry signal. This was, however, not the case. Both versions achieve practically the same deviance. As for variable selection, while the fixed intercept version also here selects more true positive variables, it comes at a cost of selecting more false positives.

6.6 Conclusion

In general, we see that FHTBoost is able to estimate FHT models which achieve good performance, on data generated from a true FHT mechanism. The performance achieved is especially high for the uncorrelated scenario, which is to be expected. As we have discussed, the variable selection has trade offs. The fixed intercept version selects more true positives, but it also selects more false positives. It does seem to be worth it, in the end, since it achieves a better test error, i.e. a lower difference of deviance. The results are also encouraging in the highly correlated scenario. Since this is a much more realistic scenario, we should give more weight to the results attained in it. If so, it is difficult to make a conclusive statement on which version is better, as the results are very similar in the highly correlated scenario. However, to narrow down the scope of

the next chapter, we will decide to only use the fixed intercept version, as we at least cannot say that the changing intercept bersion is better.

Chapter 7

Application on real data

In the simulation study in the previous chapter, we contrasted the fixed intercept version and the updating intercept version of FHTBoost. In general, FHTBoost seems to provide a decent model for correlated, realistic survival data, with average difference of deviance much smaller than 0, meaning that including covariates in the model increases its fit to test data. We now evaluate the performance of FHTBoost on a real data example. Specifically, we consider data from Oberthuer et al. (2008), consisting of data from patients diagnosed with neuroblastoma. The dataset includes a relatively small number of patients, with information on two clinical covariates and around 10 000 gene expressions. We use FHTBoost on this data to estimate an FHT model, and assess the model fit with deviance. Finally, we compare the predictive power of FHTBoost to CoxBoost via the Brier score (section 5.3), which is a method to estimate a Cox regression model using a boosting framework (see, e.g., Binder and Schumacher (2008)). We first give a description of the data set.

7.1 Description of the neuroblastoma data set

We consider survival data from Oberthuer et al. (2008), consisting of patients diagnosed with neuroblastoma. Neuroblastoma is a malignant pediatric tumor that accounts for about 8% of all childhood cancers. One of the hallmarks of the disease is its contrasting biological behavior, which results in diverse clinical courses ranging from spontaneous regression to rapid and fatal tumor progression despite intensive treatment. In recent years, several markers have been reported to offer valuable prognostic information. These markers are routinely determined by the current German neuroblastoma clinical trial NB2004 to stratify patients into groups of high risk (50%) or low risk (50%) of disease. Therapeutic strategies vary according to these risk categories, and they range from a wait-and-see approach for those in the low risk group to intensive treatment for the high-risk group. The risk categories are no silver bullet for predicting survival. On the contrary, common clinical experience suggests that such risk classification is still suboptimal for a substantial number of patients.

Originally, the data consist of two separate data sets: A larger training set, collected in Germany, and a smaller test set, collected in several countries. The training set consists of 256 patients of the German Neuroblastoma Trials NB90-NB2004, where the patients were diagnosed between 1989 and 2004. The

test set is an independent set of 120 patients from national trials in several countries (including 29 from Germany). Due to a low number of events in the NB2004 low risk group, and following Bøvelstad et al. (2009), we merge the "training set" and the "test set" into one data set. Hence, in total, the data consists of 362 patients suffering from neuroblastoma. There are 9978 gene expressions measurements, comprised of those measurements that are in probes from both the "training set" and the "test set." From each patient, we have information on their risk group according to the current German neuroblastoma trial, as well as the patient's age at diagnosis. In this data set, 89 out of the 362 observations have missing values. We remove all of these observations, and are left with a data set of 273 individuals with full observations. Finally, for each individual we have the possibly censored event time, which in the case of an event is the time to a fatality, in years from diagnosis. The median follow-up time without a fatal event was 4.14 years. 42 of the 273 children experienced a fatal event within the follow up, which is a proportion of 31.5%. Of these, 86 children were classified as having high-risk, while the remaining 187 were not. The risk indicator is coded as a binary variable, where low risk is coded as 0, and high risk as 1. We will now briefly discuss the models we intend to estimate on this data set, and how we will evaluate their performance.

7.2 Methods

7.2.1 Comparing the full FHT model to clinical-only and genetic-only FHT models

Bøvelstad et al. (2009) analysed the neuroblastoma data and compared different statistical methods to combine clinical and genomic data in Cox models. As mentioned in subsection 2.5.4, the FHT model lends itself easily to combining genomic and clinical data, and this holds for FHTBoost as well. There is also a straightforward way to only use one kind of covariates, i.e., only clinical covariates or only genetic covariates. To do this, we can use the cyclical version of the algorithm, where we boost both parameters in each step, but each has its own tuning parameter. This lets us fix the number of boosting steps of the parameter not to be boosted to 0. In other words, we make a genomic version of FHTBoost, or y_0-only version of FHTBoost, by only boosting the initial level y_0. This version of FHTBoost has $m_{\text{stop},1}$, corresponding to μ, fixed at 0, while we perform cross-validation in the usual way to find the optimal $m_{\text{stop},2}$, corresponding to y_0. Similarly, the clinical version fixes $m_{\text{stop},2}$ at 0, and tunes the other, $m_{\text{stop},1}$, corresponding to μ. In this way, we can compare the performance of our model across the genetic and clinical data, in a similar way as in Bøvelstad et al. (2009).

7.2.2 Comparing the predictive power of an FHT model with the Cox model

To properly assess the predictive power of our estimated FHT models, we need to compare it to state of the art methods. As we have discussed the Cox model earlier (see subsection 2.2.2), and we have implemented a boosting algorithm for the FHT model, we find it appropriate to use *CoxBoost* (Binder, 2013). It is an R package for estimating boosted additive predictors for the Cox model.

There exists a gradient boosted Cox estimation method in *mboost*, but its cross-validation procedure in was broken at the time of this thesis. *CoxBoost* uses likelihood based boosting (Tutz and Binder, 2006), which is a similar boosting method that arrives at very similar Cox estimates. In likelihood boosting, the tuning parameter is λ, and while it is not directly comparable to the step length ν in gradient boosting algorithms, we set

$$\lambda = \frac{N(1-\nu)}{\nu}, \tag{7.1}$$

{eq:lambda-nu}

as suggested in De Bin (2016). The size of λ, like the step length in gradient boosting, is not very important. As long as it is roughly the appropriate size, it does not have a large effect on parameter estimates. Here N is the number of individuals in the data set on which the estimator is applied, and ν is the step length mentioned in the various boosting algorithms in Chapter 3, which we by convention always set to 0.1.

In the Cox model, we include all covariates when modeling the hazard function, i.e. for this neuroblastoma data set it estimates a model of the form

$$\hat{\boldsymbol{\eta}}_{\text{CB}} = \underbrace{\hat{\beta}_{1,1}\boldsymbol{x}_1 + \hat{\beta}_{1,2}\boldsymbol{x}_2}_{\text{clinical}} + \overbrace{\hat{\beta}_{2,1}\boldsymbol{z}_1 + \ldots + \hat{\beta}_{2,9978}\boldsymbol{z}_{9978}}^{\text{genomic}},$$

since there are two clinical covariates and 9978 genomic ones. In the standard boosted Cox model, all parameters are estimated in a penalized manner. Thus there is a chance of the clinical data not being included in a final estimated model, like for FHTBoost. A gene covariate will not be treated any differently than a clinical covariate, but that is something we may want, as our FHT model does treat them differently. There is also an option in *CoxBoost* for specifying mandatory covariates. These covariates will not be penalized, and hence they will be included in the final model. We therefore in this chapter, in addition to the regular CoxBoost model, estimate a mandatory CoxBoost model, where we treat clinical covariates as mandatory. To estimate the survival probabilities given these models, we use the method discussed in subsection 2.2.3. We now discuss the results from one split of the data into training and test sets.

7.3 A single split of the data set

Due to the lack of data, Bøvelstad et al. (2009) generated 50 random splits of training and test sets from the merged data set. We do the same, only we generate 100 random splits. This gives a more accurate estimate of the model performance, as we average out some of the variance which arises from splitting a data set of with a small sample size. First, however, we report the analysis of one single (the first) split of train and test data, to show how to interpret the results. The data are split in a training set (2/3) and test set (1/3), stratified by the event status. The training set consists of 182 patients, where 28 are observed events. The test set consists of 91, where 14 are observed events. As usual, we estimate the models on a training set, and evaluate their performance on a test set.

Figure 7.1: Repeated 5-fold cross validation on training set generated from neuroblastoma data set (Oberthuer et al., 2008). The grey lines refer to the cross validation error for a single 5-fold CV implementation, the black line is the average of 10 such replications. The red vertical line highlights the best value for the number of boosting steps.

fig: neuroblastoma-cv

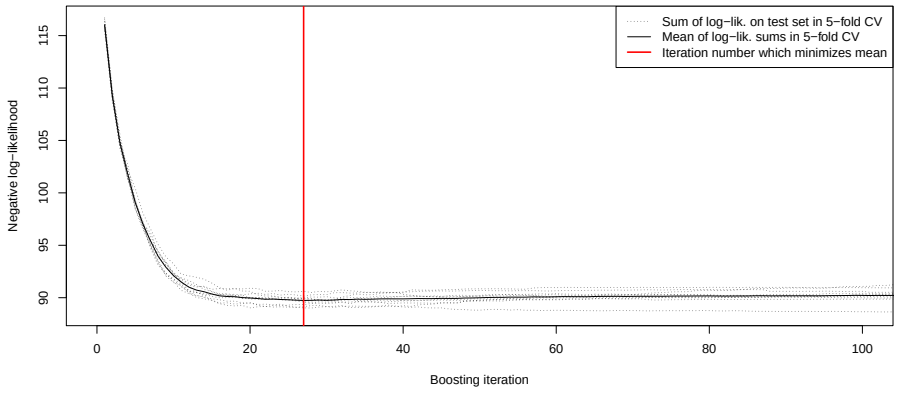


Table 7.1: FHT null model additive predictors estimated on a single split of the neuroblastoma data set (Oberthuer et al., 2008)

tab: neuroblastoma-intercepts

Intercept	Value
$\hat{\beta}_0$	0.666
$\hat{\gamma}_0$	0.332

7.3.1 Cross-validation on training set

As has been shown previously, cross-validation should be repeated with different division of folds, as this reduces variance in the estimator of m_{stop}. We therefore perform a (10 times) repeated 5-fold cross validation to find the optimal number of iterations. The resulting test errors are shown in Figure 7.1. In this example the boosting algorithm, in contrast to the original AdaBoost, eventually overfits, so it is important to select the number of boosting steps by cross-validation. Note the impact of running the cross-validation in a repeated fashion: Each dotted gray line in the figure is the sum of the negative log-likelihood of a model trained on 4 folds and applied to the last fold, as a function of the iteration number m, i.e. the cross-validated test error. We would not have selected the same m_{stop} in all of these 10 single 5-fold cross-validations, because the minimizing m for each line ranges from 22 to 50. The solid black line is the mean of these 10 gray lines, and it is this line that we choose the minimizer of. We find $m_{\text{stop}} = 27$ to be optimal, i.e., the minimizing iteration number, shown as the red vertical line in Figure 7.1. We run the FHTBoost algorithm with a fixed intercept for m_{stop} iterations, resulting in an estimated model. We first examine the model parameters.

7.3.2 Interpretation of estimated null model

In an FHT context with a Wiener process, a null model, i.e. without covariates, means a model with initial level

$$\hat{y}_0^{[0]} = \exp\left(\hat{\beta}_0\right)$$

and drift

$$\hat{\mu}^{[0]} = \hat{\gamma}_0.$$

Note that since we are using the fixed intercept version of FHTBoost, the null model will be the same for the three types of the FHT model we are considering here. The estimated intercepts are reported in Table 7.1. The estimated intercept for the gene data, $\hat{\beta}_0$, is 0.666. Remember that the vector $\boldsymbol{\beta}$ corresponds to the initial level y_0 of the health process, through the log link function. The null model, without any covariate effects, therefore has a y_0 of $\exp(0.666) = 1.946$. The intercept γ_0 corresponding to the clinical data is estimated to be 0.332. Since these are related by identity link, the drift μ is also 0.332. This means that the health process with the FHT interpretation that arises from our estimation is a Wiener process with a relatively small initial level of 1.946, and with a positive drift of 0.332. Since we are looking at when, if ever, a neuroblastoma patient will have a fatal event after diagnosis, a relatively small initial level of the Wiener process makes sense. It is the health level of a patient *at* its diagnosis. The health of such a child is presumably quite precarious, as neuroblastoma is a malignant cancer. It also makes sense that the drift μ is positive, as this means the health level of a child will on average increase after it is diagnosed. Since the individuals in the study are young children, and we are looking at a timeframe that does not comprise the length of a typical human life, the health level of most children should indeed increase after diagnosis. We recall from section 2.3.3 that a Wiener process $W(t)$ has variance equal to its time t. The large variability of the process relative to its starting point means there is still a chance of dying of the cancer. The resulting health process is

$$Y(t) = 1.946 + W(t) \cdot 0.332t,$$

where $W(t) \sim N(0, \sqrt{t})$, i.e.,

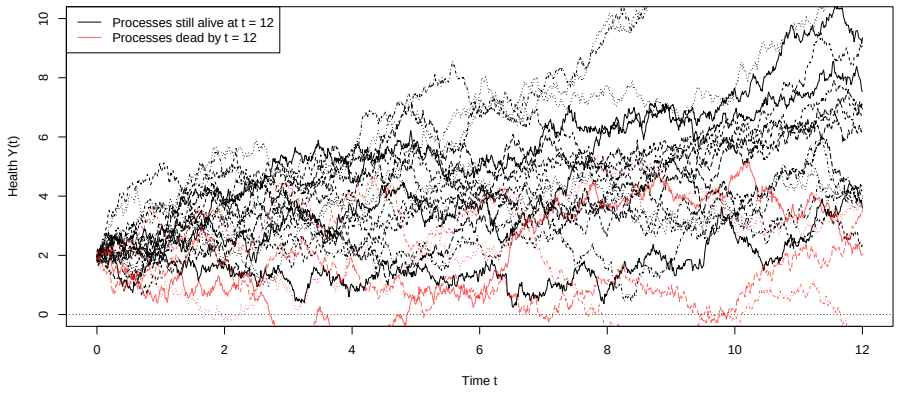
$$Y(t) \sim N(1.946 + 0.332t, \sqrt{t}),$$

where $N(\mu, \sigma)$ is the normal distribution with mean μ and standard deviance σ. To get a feeling of the variability of $Y(t)$, and potential trajectories of such a process, we plot 20 realizations of this process in Figure 7.2. Of these particular 20 processes, 4 processes at some point go below 0. These are given a red color in figure 7.2. In the FHT interpretation, then, these health processes would cause a fatal event of the child. To get a better estimate of the proportion of health processes causing events, we sample more processes. Of 10000 sampled processes, 2309 went below 0 within $t = 12$, i.e., a proportion of 0.769 did not experience a fatal event. In section 2.3, specifically in equation (2.22), we gave the probability of an IG FHT lifetime not ending, i.e., the cure rate. We will have a non-zero cure rate if the drift is positive, like in our estimated null model. We calculate its cure rate, obtaining

$$\begin{aligned}\Pr(T = \infty) = 1 - \Pr(T < \infty) &= 1 - \exp\left(-2 \cdot y_0^{[0]} \cdot \mu^{[0]}\right) \\ &= 1 - \exp(-2 \cdot 1.946 \cdot 0.332) = 0.725,\end{aligned}$$

Figure 7.2: Curves for 20 randomly generated Wiener processes with parameters $y_0 = 1.946$ and $\mu = 0.332$, corresponding to the estimated null model from the neuroblastoma data set (Oberthuer et al., 2008).

fig: neuroblastoma-wien



meaning about one in four children are estimated to die from neuroblastoma. For this training set, 28 out of 182 are observed events, meaning 0.154 have experienced a fatal event, i.e. if a proportion of 0.846 survived the cancer while under study. This is quite near the cure rate obtained from the estimated null model. Note that this is with a medium follow-up on patients of 4.5 years, while we simulated the paths of all processes for 12 years. As mentioned, and as clinicians experience, the survival probability of patients vary, even among those specified as high-risk. We therefore must look at estimated covariates to see if we can understand this variability.

7.3.3 Estimated covariate effects

The estimated covariate effects are reported in Table 7.2 ($\hat{\boldsymbol{\gamma}}$) and Table 7.3 ($\hat{\boldsymbol{\beta}}$), where the estimates in the full model are reported in the middle columns. We first observe that the boosting algorithm has included both clinical covariates into the model, i.e. the age at diagnosis and the risk indicator. The full model has also included 8 genes for this training set. Remember that we centered each column of the covariate matrices, such that the mean across all individuals is (approximately) 0 for each covariate j. For the gene expressions, this is not particularly important with regards to interpretation, as the scale of these do not lend themselves easily to interpretation. However, to properly consider the interpretation of the estimated parameters for to the clinical measurements, we should consider these on their original scale. For example, the covariate corresponding to risk, namely γ_1, is originally either 0 or 1, depending on the covariate. After standardizing, these covariates are -0.677 and 1.472, respectively. The most striking result here is the large size of the estimated parameter corresponding to risk, namely -0.259. We calculate the drift parameter for those individuals designated as high-risk, and it is

$$\mu^{[0]}_{\text{high-risk}} = 0.332 - 0.259 \cdot 1.472 = -0.049,$$

Table 7.2: Results of estimated clinical coefficients on neuroblastoma data (Oberthuer et al., 2008).

tab:oberthuer-gamma

Clinical covariate	$\hat{\gamma}_j$ (full)	$\hat{\gamma}_j$ (clinical only)
Risk	-0.259	-0.371
Age	-0.039	0

whereas for those designated as low and intermediate risk, it is

$$\mu^{[0]}_{\text{low-risk}} = 0.332 - 0.259 \cdot -0.677 = 0.507.$$

The drift for those with high risk is negative, while it is quite positive, for low risk. Since the age column is standardized, these two drift covariates are the mean drift parameters for each of the groups. We may check that this holds true for all individuals within the risk groups. The effect of age is negative, which means that there is some downward effect on the drift as the child's age increases. This negative effect could also potentially lead to low-risk individuals having an estimated negative drift, if the child is sufficiently old. This is, however, not the case. The effect of age is so small as to not impact the sign of the drift. Because the maximum standardized age is 4.193, the age contribution to drift is bounded below by

$$-0.039 \cdot 4.193 = -0.164.$$

Similarly, there are no high-risk individuals for which the drift will be positive, since the minimum standardized age is -0.724, and so the effect of age is bounded above by

$$-0.039 \cdot -0.724 = 0.028.$$

Hence, unfortunately, this means that the model predicts that all high-risk children will eventually die from of neuroblastoma cancer. This seems to resonate with the fact that these children are indeed characterized as having a high risk of recurrence. Those not designated as high-risk, on the other hand, have a non-zero probability of not experiencing recurrence. We calculate it using the formula seen previously, namely

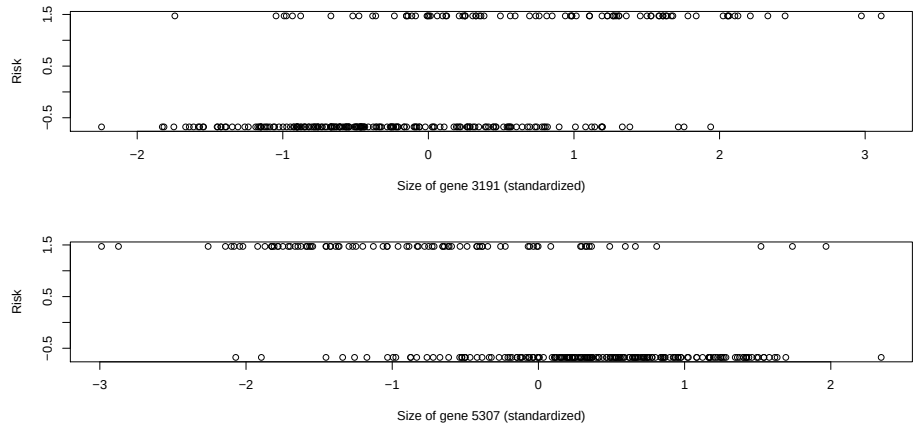
$$1 - \exp\left(-2 \cdot y_0^{[0]} \cdot \mu^{[0]}_{\text{low-risk}}\right) = 1 - \exp\left(-2 \cdot 1.946 \cdot 0.507\right) = 0.861.$$

We should therefore expect, based on the estimated model, that almost all of the low-risk patients should recover. Briefly, we also observe that the estimated parameter for the risk covariate is slightly larger, in absolute terms, for the clinical-only FHT model, at -0.371. We should also consider the gene covariates.

We observe that 7 genes have been selected in 27 boosting iterations (see the $\hat{\beta}_j$ column in Table 7.3). Some effects are estimated to be positive, some to be negative, but all are roughly the same size in absolute terms. Again, the gene expression measurements have been centered and scaled. However, a larger parameter does not necessarily mean that the effect of this gene is in general larger than others, as it still depends somewhat on the distribution of these. We recall e.g. from the simulation chapter that the effect of a change in β_j is multiplicative in $\exp(\beta_j)$, and not additive. We might for example look at gene

Figure 7.3: Scatterplots of genes 3191 (top) and 5307 (bottom) with the risk indicator for the neuroblastoma data set.

fig:gene-corr



5527. The estimated effect of it is $\hat{\beta}_{5527} = -0.096$. The maximum observation of this gene after standardization is 2.664. For a hypothetical observation which has mean values, and thus no effect, for all other genes, but with max effect of this gene, its $\hat{y}_0$ will be

$$\hat{y}_0 = \hat{y}_0^{[0]} \cdot \exp(-0.096 \cdot 2.664) = 1.946 \cdot 0.774 = 1.506,$$

whereas the null model has an initial level of 1.946. Hence this gene has quite a large effect on the initial level of this hypothetical observation, and hence its estimated probability of dying increases.

Finally, we consider the estimated gene effects of the genomic-only model, reported in the rightmost column of Table 7.3. Only four genes are selected in the model, and two of these are not selected in the somewhat larger full model, namely gene 3191 and 5307. It turns out that these are highly correlated with the risk indicator. For gene 3191 the correlation is 0.532, and for 5307 it is -0.597. The positive correlation for gene 3191 means that if a child has a large measured gene effect, the likelihood of it being diagnosed as high risk is larger than if it has a mean effect of this gene. Conversely, since the correlation for gene 5307 is highly negative, it is likely that a child with a high value of this gene has a low risk indicator, compared to if it has a mean effect of this gene. We show a scatterplot of these genes and the risk indicator in Figure 7.3.

7.3.4 Measurement results on the test set

We now examine the results of the estimated models on the test set. The test set in this split consists of 91 individuals, of which 14 have experienced recurrence.

7.3.4.1 Difference of deviance

We calculate the difference in deviance for the estimated FHT model. As explained in Section 5.1, the difference of deviance is twice the difference between the log-likelihood for an estimated model which includes covariates,

Table 7.3: Results of estimated gene coefficients on neuroblastoma data (Oberthuer et al., 2008). tab:oberthuer-beta

Gene j	$\hat{\beta}_j$ (full)	$\hat{\beta}_j$ (genomic only)
Gene 1447	-0.064	-0.093
Gene 2442	0.038	0
Gene 2783	-0.024	0
Gene 3191	0	-0.093
Gene 5307	0	0.046
Gene 5527	-0.096	-0.201
Gene 5725	-0.011	0
Gene 6901	0.013	0
Gene 7523	-0.023	0

Table 7.4: Difference of deviance with FHTBoost with a fixed intercept, on a single split of the neuroblastoma data (Oberthuer et al., 2008). tab:deviances

Boosting type	Difference of deviance
Full	-108.9
Clinical (y_0) only	-26.2
Genomic (μ) only	-129.7

and a model without covariates. The performance of a model is good when the difference in deviance is small (i.e. negative with a large absolute value). We obtain a difference of deviance of -108.9 for the full FHT model. It means that under the assumption that the FHT model is true, a lot of variation is explained by covariates. For this test set, the full model is in fact beaten by the genomic model, with difference of deviance of -108.9 and -129.7, respectively. The clinical model is worse, but still better than the null model, with a difference of deviance at -26.2.

7.3.5 Brier score

We now calculate Brier scores (Section 5.3) on each observation in the test set with each FHT model, see Figure 7.4 for a graphic display of these. Interestingly, the full FHT model is quite consistently better than the genomic model, i.e. it is better per observation in the Brier score, even though it achieves a worse difference of deviance. Furthermore, the Brier score of the full FHT model is almost equal to the clinical one in some parts, except in the middle, from 3 to 7 years, where the full model is somewhat better. It is perhaps easiest to compare the performance if we have a number, and we get that by calculating the integrated Brier score, explained in subsection 5.3.3, and which is effectively the average Brier score over a given range of time, in this case the times from the entire test set. We find that the integrated Brier score for the full FHT model is 0.051. The genomic model attains an integrated Brier score of 0.085, while the clinical attains 0.057. For reference, the estimated FHT null model achieves 0.122, which is almost the double.

A comparison of the Brier score for the full FHT model and the regular Cox model can be seen in Figure 7.5. We observe that the Cox model performs

Figure 7.4: Brier scores for FHT models.

fig:brier-FHT

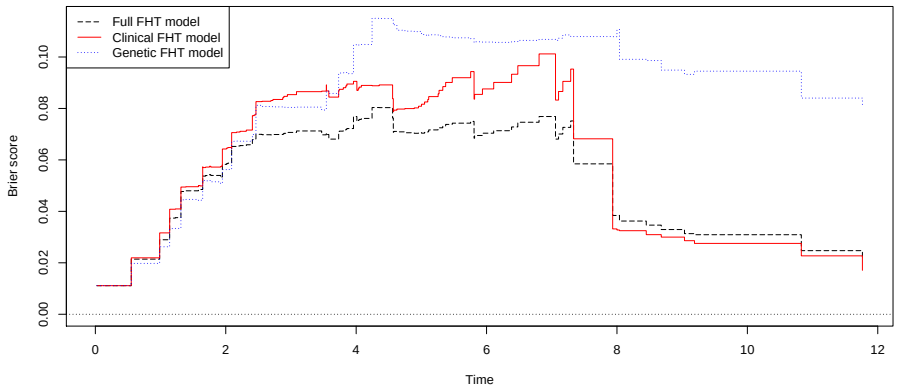
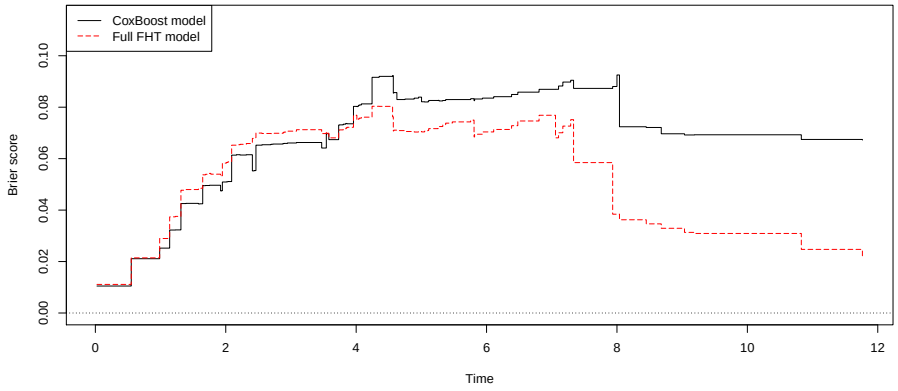


Figure 7.5: Brier scores on a single split of the test set of the neuroblastoma data, for an estimated CoxBoost survival model and a full estimated FHT model.

fig:brier-cox-both



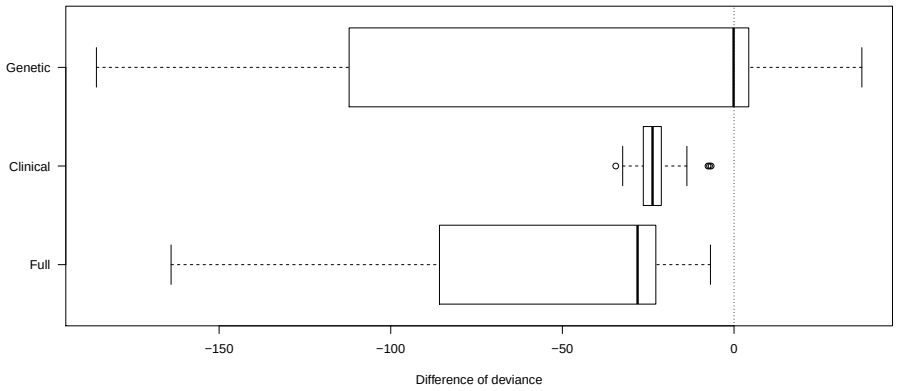
slightly better at the beginning, but at the end, the FHT model has better predictive performance. Note, however that large part of the additional error of the Cox model refers to the right part of the error curve (see Figure 7.4), where the computations are more unstable due to the reduced number of events (larger proportion of censored data). The estimated CoxBoost model attains 0.067, and so it is beaten by the full FHT model. However, the mandatory Cox model achieves an integrated Brier score of 0.044.

7.4 Results of 100 train/test splits

As mentioned above, we generate 99 additional splits into training and test sets, for a total of 100. We proceed in the same manner on the single split, i.e. estimating parameters by first finding an optimal number of iterations

Figure 7.6: Boxplot for difference in deviance for different variants of the FHT model.

fig: neuroblastoma-deviances



m_{stop}, and then running the boosting algorithm for that amount of iterations. Finally, we use the estimated model on the test set and calculate the evaluation measures.

7.4.1 Difference of deviance in FHT models

Figure 7.6 shows a boxplot of the difference of deviance across all 100 splits of training and test sets. The main measure of interest is the mean for each, and it is -51.0 for the full model, -23.4 for the clinical model, and -40.3 for the genomic model. The full FHT model has the best mean value here, which should be expected, as it can include both clinical and genomic covariates.

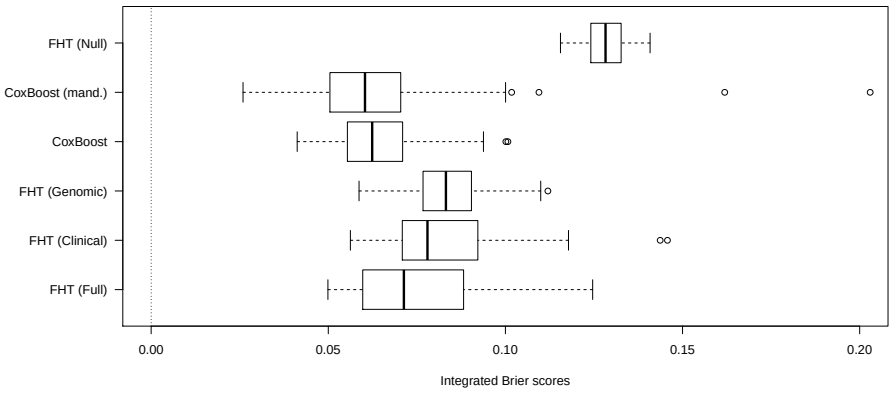
Median both clinical genetic -28.0985864 -23.7328969 -0.1257181 All of these also on

7.4.2 Integrated Brier scores

We now calculate the integrated Brier scores for all 100 splits, for all of the mentioned methods. A boxplot can be seen in Figure 7.7. The full FHT model performs best of the FHT models, which again should be the case, as it can use all covariates. We also observe that the FHT null model is on average a lot worse than all other models at predicting survival. It is, however, beaten by both Cox models, but only slightly. This is encouraging, as these results show that regression with FHT models can compete with Cox regression, also on realistic and high-dimensional data. Mean for full FHT model is 0.074. Mean for Cox model is 0.064, Cox mandatory 0.069. Clinical FHT model is 0.083. Genetic 0.084. Null model FHT is 0.128

Figure 7.7: Boxplot of integrated Brier scores.

fig: neuroblastoma-integrated-brier



Chapter 8

Discussion and future work

sec:discussion

In this thesis, we have looked at problems in survival data, and specifically first hitting time models. While Cox regression is by far the most popular method used to estimate survival data models, it has shortcomings that we have discussed. One of them is the fact that it relies on a proportional hazards assumption, which does not hold in a variable selection setting, which is almost always the case when we perform regression with high-dimensional data. The Cox model does still, however, work well in practice, and has good predictive power. First hitting time (FHT) models are flexible alternatives that do not rely on the proportional hazards assumption. As of the time of writing for this thesis, no methods for FHT models exist to estimate parameters in a high-dimensional setting. We have therefore discussed ways of estimating models that work well in such a setting. In particular, we have discussed gradient boosting (Friedman, 2001), both methods for estimating one parameter, and extensions to several parameters.

Our goal with the thesis work was therefore to combine FHT models and gradient boosting. To this goal, we have developed an algorithm for fitting an FHT model with linear additive predictors. The estimation algorithm works as follows. It starts by initializing the additive predictors to intercepts, specifically the intercepts that maximize the log-likelihood of the training set. We then perform iterations where we in each step include a regularized linear least squares function in *one* of the covariates and *one* of the parameters, namely the combination of covariate and parameter which leads to the largest increase in the log-likelihood function. The algorithm was implemented from scratch as a package which we called *FHTBoost*, and it is freely available for download at `https://github.com/vegarsti/fhtboost`. It can be installed directly in R by using a command in the DevTools R package (Wickham et al., 2018) called `install_github`, namely `install_github("vegarsti/fhtboost")`.

In the boosting algorithm, we used component-wise linear learners without intercepts. We developed two versions of FHTBoost, one with a fixed intercept, and one where the intercept is treated as a nuisance parameter, and is optimized in each iteration until the boosting is stopped. In a simulation study, we found that both versions manage to select informative variables and shrink parameter sizes, and, importantly, increase model fit while doing so. In an uncorrelated scenario, the fixed intercept version achieves markedly better deviance on a test set. However in a correlated and realistic scenario, the two versions achieve

very similar performance. In all cases, the mean deviance is well below zero. To limit the scope of the analysis, we chose to use only the fixed intercept version in the next section, which looked at a neuroblastoma data set (Oberthuer et al., 2008), which is a realistic data set consisting of measurements of 9978 genes and two clinical covariates. When used on this data set, FHTBoost always estimates a model where covariates increase the log-likelihood of the model, on a test set, meaning the model fit is better than in the case of no covariates. The models estimated with FHTBoost have good predictive power, achieving a mean Brier score of 0.074. The mean for the regular Cox model is 0.064, while for a mandatory version of Cox, which did not penalize the clinical covariates, the mean was 0.069. In other words, the FHTBoost model achieves comparable predictive power as state of the art Cox models, but init did not outperform it.

There are several interesting directions for further work. One is to apply *FHTBoost* on other real-life high-dimensional survival data sets. This would allow for a broader assessment of its usefulness and predictive power on real-life problems. One of the strengths of gradient boosting is that it is easy to use in combination with different component-wise base learners. In this thesis, we have only considered linear base learners. It would be interesting to include nonlinearity into these, which could be done by e.g. using regularized splines of one dimension as base learners. Another interesting direction of further work would be to incorporate the FHT model into the existing ecosystem of gradient boosting packages in R. This ecosystem is based on the package *mboost* (Hothorn et al., 2018). The framework used for FHT regression fits into the GAMLSS framework. It should therefore be possible to use the *gamboostLSS* (Mayr et al., 2012a; Hofner et al., 2018) package. There also exists an extension for fitting GAMLSS models to censored data, *gamboostLSS.cens* (Stasinopoulos et al., 2018). It should be possible to include our parameterization of the inverse Gaussian here. By including our inverse Gaussian parameterization with its log-likelihood and derivatives, it should be possible to use the entire library of base learners in *mboost*.

Appendices

Appendix A

Appendix 1: Differentiating the IG FHT

appendix

First we have the likelihood,

$$L(y_0,\mu)=\prod_{i=1}^{n}\left(\frac{y_0}{\sqrt{2\pi\sigma^2 t_i^3}}\exp\left[-\frac{(y_0+\mu t_i)^2}{2\sigma^2 t_i}\right]\right)^{\delta_i} \times\left[1-\Phi\left(-\frac{y_0+\mu t_i}{\sqrt{\sigma^2 t_i}}\right)-\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t_i-y_0}{\sqrt{\sigma^2 t_i}}\right)\right]^{1-\delta_i}, \tag{A.1}$$

{eq:fht-lik}

with respect to parameters μ, and y_0. First, note that for any cumulative distribution function F that is symmetric around 0, and for $x\in\mathbb{R}$,

$$F(x)=1-(1-F(x))=1-F(-x), \tag{A.2}$$

and so in particular,

$$\Phi(x)=1-(1-\Phi(x))=1-\Phi(-x), \tag{A.3}$$

and thus we can rewrite (A.1) as

$$L(y_0,\mu)=\prod_{i=1}^{n}\left(\frac{y_0}{\sqrt{2\pi\sigma^2 t_i^3}}\exp\left[-\frac{(y_0+\mu t_i)^2}{2\sigma^2 t_i}\right]\right)^{\delta_i} \times\left[\Phi\left(\frac{y_0+\mu t_i}{\sqrt{\sigma^2 t_i}}\right)-\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t_i-y_0}{\sqrt{\sigma^2 t_i}}\right)\right]^{1-\delta_i}. \tag{A.4}$$

{eq:fht-lik-proper}

It is easier to work with the log likelihood, so we take the log of (A.4) and get

$$l(y_0,\mu)=\sum_{i=1}^{n}\delta_i\left(\ln y_0-\frac{1}{2}\ln\left(2\pi\sigma^2 t_i^3\right)-\frac{(y_0+\mu t_i)^2}{2\sigma^2 t_i}\right) +(1-\delta_i)\ln\left(\Phi\left(\frac{\mu t_i+y_0}{\sqrt{\sigma^2 t_i}}\right)-\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t_i-y_0}{\sqrt{\sigma^2 t_i}}\right)\right) \tag{A.5}$$

{eq:fht-loglik}

To make things easier, let us introduce some intermediate functions here. Let

$$\ln f_i(y_0,\mu)=\ln y_0-\frac{1}{2}\ln\left(2\pi\sigma^2 t_i^3\right)-\frac{(y_0+\mu t_i)^2}{2\sigma^2 t_i} \tag{A.6}$$

and

$$S_i(y_0,\mu) = \Phi\left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}}\right) - \exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right). \tag{A.7}$$

So we get

$$l(y_0,\mu) = \sum_{i=1}^{n} \delta_i \ln f_i(y_0,\mu) + (1-\delta_i)\ln S_i(y_0,\mu) \tag{A.8}$$

Thus we see that the partial derivatives are

$$\frac{\partial}{\partial y_0} l(y_0,\mu) = \sum_{i=1}^{n} \delta_i \frac{\partial}{\partial y_0} \ln f_i(y_0,\mu) + (1-\delta_i)\frac{\frac{\partial}{\partial y_0} S_i(y_0,\mu)}{S_i(\theta)} \tag{A.9}$$

and

$$\frac{\partial}{\partial \mu} l(y_0,\mu) = \sum_{i=1}^{n} \delta_i \frac{\partial}{\partial \mu} \ln f_i(y_0,\mu) + (1-\delta_i)\frac{\frac{\partial}{\partial \mu} S_i(y_0,\mu)}{S_i(\theta)} \tag{A.10}$$

We take these one by one

$$\frac{\partial}{\partial y_0} \ln f_i(y_0,\mu) = \frac{1}{y_0} - \frac{y_0 + \mu t_i}{\sigma^2 t_i} \tag{A.11}$$

$$\frac{\partial}{\partial \mu} \ln f_i(y_0,\mu) = -\frac{y_0 + \mu t_i}{\sigma^2} \tag{A.12}$$

$$\begin{aligned}\frac{\partial}{\partial y_0} S_i(y_0,\mu) = {} & \frac{1}{\sqrt{\sigma^2 t_i}}\phi\left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}}\right) + \frac{2\mu}{\sigma^2}\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right) \\ & + \frac{1}{\sqrt{\sigma^2 t_i}}\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right)\end{aligned} \tag{A.13}$$

$$\begin{aligned}\frac{\partial}{\partial \mu} S_i(y_0,\mu) = {} & \frac{t_i}{\sqrt{\sigma^2 t_i}}\phi\left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}}\right) + \frac{2y_0}{\sigma^2}\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right) \\ & - \frac{t_i}{\sqrt{\sigma^2 t_i}}\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right)\end{aligned} \tag{A.14}$$

Hence

$$\begin{aligned}&\frac{\partial}{\partial y_0} l(y_0,\mu) \\ &= \sum_{i=1}^{n}\left(\delta_i\left(\frac{1}{y_0} - \frac{y_0+\mu t_i}{\sigma^2 t_i}\right) + (1-\delta_i)\left[\frac{1}{\sqrt{\sigma^2 t_i}}\phi\left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}}\right) + \frac{2\mu}{\sigma^2}\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right)\right.\right. \\ &\left.\left. + \frac{1}{\sqrt{\sigma^2 t_i}}\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right)\right]\left[\Phi\left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}}\right) - \exp(-\frac{2y_0\mu}{\sigma^2}\Phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right)\right]^{-1}\right)\end{aligned} \tag{A.15}$$

and

$$\frac{\partial}{\partial\mu}l(y_0,\mu)$$
$$=\sum_{i=1}^{n}\left(\delta_i\left(-\frac{y_0+\mu t_i}{\sigma^2}\right)+(1-\delta_i)\left[\frac{t_i}{\sqrt{\sigma^2 t_i}}\phi\left(\frac{\mu t_i+y_0}{\sqrt{\sigma^2 t_i}}\right)+\frac{2y_0}{\sigma^2}\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t_i-y_0}{\sqrt{\sigma^2 t_i}}\right)\right.\right.$$
$$\left.\left.-\frac{t_i}{\sqrt{\sigma^2 t_i}}\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\phi\left(\frac{\mu t_i-y_0}{\sqrt{\sigma^2 t_i}}\right)\right]\left[\Phi\left(\frac{\mu t_i+y_0}{\sqrt{\sigma^2 t_i}}\right)-\exp(-\frac{2y_0\mu}{\sigma^2}\Phi\left(\frac{\mu t_i-y_0}{\sqrt{\sigma^2 t_i}}\right)\right]^{-1}\right) \tag{A.16}$$

Appendix B

Appendix 2: R code

appendix2

B.1 Generate correlated gene and clinical data

code:generate-correlated-data

```
generate_clinical <- function(n.obs=200,tot.genes=10000,n.groups=100,n.clin=NULL,n.gene=NULL,mean.n.gene=15,
                        mu.g=6,sigma.g=0.65,mu.c=1,sigma.c=0.5,rho.c=0,rho.b=0,rho.g=0,phi=0.1,nu=10,tau=20) {
  # n.obs (integer) = number of observations
  # tot.genes (integer) = number of molecular predictors
  # n.groups (integer) = number of pathways (molecular predictors correlated to each other)
  # n.clin (vector) = number of clinical predictors for each group (if NULL, no clinical predictors are generated)
  # n.gene (vector) = number of molecular predictors for each group (if NULL, all the group sizes are generated randomly, if its length is smaller than n.groups the unspecified size are generated randomly as well)
  # mean.n.gene (integer) = average sizes of molecular predictors for group. Relevant only if n.gene is NULL or length(n.gene)<n.groups
  # mu.c (integer) = mean of the log-normal distribution of the clinical variables
  # sigma.c (integer) = standard deviation of the log-normal distribution of the clinical variables
  # mu.g (integer) = mean of the log-normal distribution of the genes
  # sigma.g (integer) = standard deviation of the log-normal distribution of the genes
  # rho.g (vector) = correlation within each block of genes (equal for all of them). Default (for all or only the part of the groups for which it is not specified) is 0.
  # rho.c (vector) = vector containing the correlation between the clinical predictors in each pathway
  # rho.b (vector) = vector containing the correlation between the clinical and the molecular predictors in each pathway
  # phi (integer) = standard deviation of the normal distribution modeling the multiplicative noise to the signal
  # nu (integer) = mean of the normal distribution modeling the additive noise to the signal
  # tau (integer) = standard deviation of the distribution modeling the additive noise to the signal

  require(mvtnorm)
  require(corpcor)

  if(tot.genes<n.groups) stop('The number of genes must be bigger than the number of pathways\n')
  # if the groups sizes are not provided, generate them
```

```
length.n.gene<-length(n.gene)
if(tot.genes<sum(n.gene)) stop('The number of genes must be bigger than the total number of genes in the pathways\n')
{if(is.null(n.gene)) length.n.gene<-0
  else if (length.n.gene<n.groups) warning(paste0('The length of n.gene is smaller than ',n.groups,'. The sizes of the remaining groups is generated randomly\n'))}
if (length.n.gene<n.groups)
{
  n.gene<-c(n.gene,round(rnorm(n.groups-length.n.gene,mean.n.gene,0.3*mean.n.gene)))
  n.gene[n.gene<1]<-1 # to have no empty group
}
# update the number of groups generated
sumBs<-sum(n.gene)
# if we generate more genes than those indicated in tot.gene, tell how many genes are actually generated
{if(sumBs>tot.genes)
{
  tot.genes<-sumBS
  warnings(paste0('Total number of simulated genes equal to ',sumBS,'\n'))
}
  else n.gene[n.groups+1]<-tot.genes-sumBs} # the last block contains all the genes not belonging to the first n.groups blocks
# if the correlation among genes is not specified (for all or only part of the groups), we set it to 0
if(length(rho.g)<n.groups) rho.g<-c(rho.g,rep(0,n.groups-length(rho.g)))

# check for the clinical structure and complete the lists
ifelse(is.null(n.clin),length.n.clin<-0,length.n.clin<-length(n.clin))
# check if the number of clinical groups is reasonable
if(length.n.clin>n.groups) {
  n.clin<-n.clin[1:n.groups]
  warnings(paste0('Number of clinical groups too large, only the first ', length.n.gene,' are used.\n'))
}
if (sum(rho.c) > 0) {
  n.clin[(length.n.clin+1):n.groups]<-0
}

# if correlation among clinical predictors is not specified (for all or only part of the groups) set it to 0
if(length(rho.c)<n.groups) rho.c<-c(rho.c,rep(0,n.groups-length(rho.c)))
# if correlation between clinical and molecular predictors is not specified (for all or only part of the groups) set it to 0
if(length(rho.b)<n.groups) rho.b<-c(rho.b,rep(0,n.groups-length(rho.b)))

# generate the data
Clin<-NULL
Gene<-NULL
for (i in 1:n.groups)
{
  # generate the covariance matrix
  Sigma.h<-rbind(cbind(rep(sigma.c,n.clin[i])%*%t(rep(sigma.c,n.clin[i]))*rho.c[i], # clinical part
                       rep(sigma.c,n.clin[i])%*%t(rep(sigma.g,n.gene[i]))*rho.b[i]), # clinical and molecular part, up right part of the covariance matrix
                 cbind(t(rep(sigma.c,n.clin[i])%*%t(rep(sigma.g,n.gene[i])))*rho.b[i], # clinical and molecular part, bottom left part of the covariance matrix
```

```
71                          rep(sigma.g,n.gene[i])%*%t(rep(sigma.g,n.gene[i]))*
                                rho.g[i])) # molecular part
72      diag(Sigma.h)<-c(rep(sigma.c^2,n.clin[i]),rep(sigma.g^2,n.gene[i]))
73      if(!is.positive.definite(Sigma.h)) Sigma.h<-make.positive.definite(Sigma.h
            )
74
75      # generate the data
76      tmp<-rmvnorm(n.obs,c(rep(mu.c,n.clin[i]),rep(mu.g,n.gene[i])),Sigma.h)
77      {if(n.clin[i]==0) Gene<-cbind(Gene,exp(tmp))
78        else
79        {
80          Clin<-cbind(Clin,tmp[,1:n.clin[i]])
81          Gene<-cbind(Gene,exp(tmp[,-c(1:n.clin[i])]))
82        }}
83    }
84    # the genes exceeding those clustered in the groups are generated
          uncorrelated to any other predictors
85    if(tot.genes>sumBs) Gene<-cbind(Gene,sapply((sumBs+1):tot.genes,function(i,
          mu.g,sigma.g,n.obs) exp(rnorm(n.obs,mu.g,sigma.g)),mu.g=mu.g,sigma.g=
          sigma.g,n.obs=n.obs))
86
87    # generation of the noise
88    # additive noise
89    E<-matrix(rnorm(tot.genes*n.obs,nu,tau),ncol=tot.genes,nrow=n.obs)
90    # multiplicative noise
91    M<-matrix(rnorm(tot.genes*n.obs,0,phi),ncol=tot.genes,nrow=n.obs)
92    # observations including the noise
93    Gene<-Gene*exp(M)+E
94
95    # thresholding and normalization
96    Gene[Gene<10]<-10
97    Gene[Gene>16000]<-16000
98    Gene<-log(Gene)
99
100   if (!is.null(Clin)) colnames(Clin)<-paste0('clin',1:dim(Clin)[2])
101   colnames(Gene)<-paste0('gene',1:dim(Gene)[2])
102
103   return(list(clin=Clin, gene=Gene))
104 }
```

B.2 Calculate cumulative baseline hazard in Cox

code:cumulative-baseline-hazard

```r
estimate_baseline_hazard <- function(times, delta, linear_predictors) {
  # times, delta and linear_predictors must be sorted in correct time order
  # delta is the usual observed indicator in survival analysis
  # this code calculates the estimated baseline hazard at the times given
  N <- length(times)
  jumps <- rep(0, N)
  num_events <- rep(0, N)
  denominator <- rep(0, N)
  exp_lp <- exp(linear_predictors)
  for (i in 1:N) {
    current_time <- times[i]
    at_risk_indicator <- current_time <= times
    denominator[i] <- sum(at_risk_indicator * exp_lp)
    is_event <- delta[i]
    jumps[i] <- is_event/denominator[i]
  }
  A0 <- cumsum(jumps)
  return(A0)
}
A0 <- estimate_baseline_hazard(times_test, delta_test, cox_linear_predictors)
```

Bibliography

aalen1978 Aalen, O. (1978). Nonparametric inference for a family of counting processes. *Ann. Statist.*, 6(4):701–726.

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

aalengjessing2001 Aalen, O. and Gjessing, H. K. (2001). Understanding the shape of the hazard rate: a process point of view (with comments and a rejoinder by the authors). *Statist. Sci.*, 16(1):1–22.

Akaike1998 Akaike, H. (1998). *Information Theory and an Extension of the Maximum Likelihood Principle*, pages 199–213. Springer New York, New York, NY.

bauer-kohavi Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1):105–139.

coxboost Binder, H. (2013). *CoxBoost: Cox models by likelihood based boosting for a single survival endpoint or competing risks*. R package version 1.4.

BinderSchumacher2008 Binder, H. and Schumacher, M. (2008). Adapting prediction error estimates for biased complexity selection in high-dimensional bootstrap samples. *Statistical applications in genetics and molecular biology*, 7 1:Article12.

breiman1998 Breiman, L. (1998). Arcing classifier (with discussion and a rejoinder by the author). *Ann. Statist.*, 26(3):801–849.

brier1950 Brier, G. W. (1950). Verification of Forecasts expressed in terms of probability. *Monthly Weather Review*, 78(1):1–3.

bovelstadborgan Bøvelstad, H. M. and Borgan, Ø. (2011). Assessment of evaluation criteria for survival prediction from genomic data. *Biometrical Journal*, 53(2):202–216.

bovelstad2009 Bøvelstad, H. M., Nygård, S., and Borgan, Ø. (2009). Survival prediction from clinico-genomic models - a comparative study. *BMC Bioinformatics*, 10(1):413.

buhlmann2006 Bühlmann, P. (2006). Boosting for high-dimensional linear models. *Ann. Statist.*, 34(2):559–583.

buhlmann2007 Bühlmann, P. and Hothorn, T. (2007). Boosting algorithms: Regularization, prediction and model fitting. *Statist. Sci.*, 22(4):477–505.

buhlmann-yu Bühlmann, P. and Yu, B. (2003). Boosting with the l2 loss. *Journal of the American Statistical Association*, 98(462):324–339.

caroni2017 Caroni, C. (2017). *First Hitting Time Regression Models*. John Wiley & Sons, Inc.

chhikara1988 Chhikara, R. (1988). *The Inverse Gaussian Distribution: Theory: Methodology, and Applications*. Statistics: A Series of Textbooks and Monographs. Taylor & Francis.

copas1983 Copas, J. B. (1983). Regression, prediction and shrinkage. *Journal of the Royal Statistical Society. Series B (Methodological)*, 45(3):311–354.

cox1965 Cox, D. and Miller, H. (1965). *The theory of stochastic processes*. Wiley publications in statistics. Wiley.

cox-model Cox, D. R. (1972). Regression models and life-tables. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(2):187–220.

saddlepoints Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2933–2941, Cambridge, MA, USA. MIT Press.

DeBin2016 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(2):513–531.

sauerbrei De Bin, R., Sauerbrei, W., and Boulesteix, A.-L. (2014). Investigating the prediction ability of survival models based on both clinical and omics data: two case studies. *Statistics in Medicine*, 33(30):5310–5329.

eaton-whitmore Eaton, W. W. and Whitmore, G. A. (1977). Length of stay as a stochastic process: A general approach and application to hospitalization for schizophrenia. *The Journal of Mathematical Sociology*, 5(2):273–292.

efron1975 Efron, B. (1975). Biased versus unbiased estimation. *Advances in Mathematics*, 16(3):259 – 277.

efron2004 Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004). Least angle regression. *Ann. Statist.*, 32(2):407–499.

adaboost Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*, ICML'96, pages 148–156, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

friedman2001 Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232.

bovelstad2007 Frigessi, A., Størvold, H., Bøvelstad, H., Aldrin, M., Borgan, Ø., Lingjærde, O., and Nygård, S. (2007). Predicting survival from microarray data—a comparative study. *Bioinformatics*, 23(16):2080–2087.

graf — Graf, E., Schmoor, C., Sauerbrei, W., and Schumacher, M. (1999). Assessment and comparison of prognostic classification schemes for survival data. *Statistics in Medicine*, 18(17-18):2529–2545.

hastie2007 — Hastie, T. (2007). Comment: Boosting algorithms: Regularization, prediction and model fitting. *Statist. Sci.*, 22(4):513–515.

hastie1986 — Hastie, T. and Tibshirani, R. (1986). Generalized additive models. *Statist. Sci.*, 1(3):297–310.

gam-book — Hastie, T. and Tibshirani, R. (1990). *Generalized Additive Models*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis.

ESL — Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer.

gamboostLSS-manual — Hofner, B., Mayr, A., Fenske, N., and Schmid, M. (2018). *gamboostLSS: Boosting Methods for GAMLSS Models*. R package version 2.0-1.

mboost1 — Hofner, B., Mayr, A., Robinzonov, N., and Schmid, M. (2014). Model-based boosting in R: A hands-on tutorial using the R package mboost. *Computational Statistics*, 29:3–35.

mboost2 — Hothorn, T., Buehlmann, P., Kneib, T., Schmid, M., and Hofner, B. (2010). Model-based boosting 2.0. *Journal of Machine Learning Research*, 11:2109–2113.

mboost — Hothorn, T., Buehlmann, P., Kneib, T., Schmid, M., and Hofner, B. (2018). *mboost: Model-Based Boosting*. R package version 2.9-1.

kaplan-meier — Kaplan, E. L. and Meier, P. (1958). Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, 53(282):457–481.

kearnsvaliant — Kearns, M. and 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, pages 433–444, New York, NY, USA. ACM.

kneib2013 — Kneib, T. (2013). Beyond mean regression. *Statistical Modelling*, 13(4):275–303.

kohavi — Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1137–1143, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

lachenbruch — Lachenbruch, P. A. and Mickey, M. R. (1968). Estimation of error rates in discriminant analysis. *Technometrics*, 10(1):1–11.

lancaster — Lancaster, T. (1972). A stochastic model for the duration of a strike. *Journal of the Royal Statistical Society. Series A (General)*, 135(2):257–271.

lawless2011 — Lawless, J. (2011). *Statistical Models and Methods for Lifetime Data*. Wiley Series in Probability and Statistics. Wiley.

lawless2004 Lawless, J. and Crowder, M. (2004). Covariates and random effects in a gamma process model with application to degradation and failure. *Lifetime Data Analysis*, 10(3):213–227.

leewhitmore2004a Lee, M.-L. and Whitmore, G. A. (2003). First hitting time models for lifetime data. *Handbook of Statistics*, 23:537–543.

leewhitmore2006 Lee, M.-L. T. and Whitmore, G. A. (2006). Threshold regression for survival analysis: Modeling event times by a stochastic process reaching a boundary. *Statist. Sci.*, 21(4):501–513.

leewhitmore2004 Lee, M. T., Whitmore, G. A., Laden, F., Hart, J. E., and Garshick, E. (2004). Assessing lung cancer risk in railroad workers using a first hitting time regression model. *Environmetrics*, 15(5):501–512.

maller1996survival Maller, R. and Zhou, X. (1996). *Survival Analysis with Long-Term Survivors*. Wiley Series in Child Care and Protection. Wiley.

mayr14a Mayr, A., Binder, H., Gefeller, O., and Schmid, M. (2014a). The evolution of boosting algorithms. from machine learning to statistical modelling. *Methods of Information in Medicine*, 53(6):419–427.

mayr14b Mayr, A., Binder, H., Gefeller, O., and Schmid, M. (2014b). Extending statistical boosting. an overview of recent methodological developments. *Methods of Information in Medicine*, 53(6):428–435.

gamboostlss-paper Mayr, A., Fenske, N., Hofner, B., Kneib, T., and Schmid, M. (2012a). Generalized additive models for location, scale and shape for high dimensional data—a flexible approach based on boosting. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 61(3):403–427.

mayr-hofner Mayr, A., Hofner, B., and Schmid, M. (2012b). The importance of knowing when to stop. a sequential stopping rule for component-wise gradient boosting. *Methods of Information in Medicine*, 51(2):178–186.

mayr17 Mayr, A., Hofner, B., Waldmann, E., Hepp, T., Meyer, S., and Gefeller, O. (2017). An update on statistical boosting in biomedicine. *Computational and Mathematical Methods in Medicine*, 2017:1–12.

nelson Nelson, W. (1972). Theory and applications of hazard plotting for censored failure data. *Technometrics*, 14(4):945–966.

oberthuer-data Oberthuer, A., Kaderali, L., Kahlert, Y., Hero, B., Westermann, F., Berthold, F., Brors, B., Eils, R., and Fischer, M. (2008). Subclassification and individual survival time prediction from gene expression data of neuroblastoma patients by using caspar. *Clinical cancer research : an official journal of the American Association for Cancer Research*, 14:6590–6601.

Rlang R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

gamlss Rigby, R. A. and Stasinopoulos, D. M. (2005). Generalized additive models for location, scale and shape. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 54(3):507–554.

schmid-hothorn Schmid, M. and Hothorn, T. (2008). Boosting additive models using component-wise p-splines. *Comput. Stat. Data Anal.*, 53(2):298–311.

schmid Schmid, M., Potapov, S., Pfahlberg, A., and Hothorn, T. (2010). Estimation and regularization techniques for regression models with multidimensional prediction functions. *Statistics and Computing*, 20(2):139–150.

seibold Seibold, H., Bernau, C., Boulesteix, A.-L., and De Bin, R. (2018). On the choice and influence of the number of boosting steps for high-dimensional linear cox-models. *Computational Statistics*, 33(3):1195–1215.

singpurwalla1995 Singpurwalla, N. D. (1995). Survival in dynamic environments. *Statist. Sci.*, 10(1):86–103.

gamlssR Stasinopoulos, D. M. and Rigby, R. A. (2007). Generalized additive models for location scale and shape (gamlss) in r. *Journal of Statistical Software*, 023(i07).

gamlsscens Stasinopoulos, M., Rigby, B., and Mortan, N. (2018). *gamlss.cens: Fitting an Interval Response Variable Using 'gamlss.family' Distributions*. R package version 5.0-1.

thomas2018 Thomas, J., Mayr, A., Bischl, B., Schmid, M., Smith, A., and Hofner, B. (2018). Gradient boosting for distributional regression: faster tuning and improved variable selection via noncyclical updates. *Statistics and Computing*, 28(3):673–687.

lasso Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288.

gamboost Tutz, G. and Binder, H. (2006). Generalized additive modeling with implicit variable selection by likelihood-based boosting. *Biometrics*, 62(4):961–971.

whitmore1975 Whitmore, G. A. (1975). The inverse gaussian distribution as a model of hospital stay. *Health services research*, 10:297–302.

whitmore1986 Whitmore, G. A. (1986). First-passage-time models for duration data: Regression structures and competing risks. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 35(2):207–219.

whitmore1995 Whitmore, G. A. (1995). Estimating degradation by a wiener diffusion process subject to measurement error. *Lifetime Data Analysis*, 1(3):307–319.

devtools Wickham, H., Hester, J., and Chang, W. (2018). *devtools: Tools to Make Developing R Packages Easier*. R package version 2.0.1.

threg Xiao, T., Whitmore, G., He, X., and Lee, M.-L. (2015). The r package threg to implement threshold regression models. *Journal of Statistical Software, Articles*, 66(8):1–16.