A 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

Empty.

Acknowledgements

Empty.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and outline of the thesis

In this thesis, we work with boosting for regression in the first hitting time model. First hitting time is a model in survival analysis which serves as an alternative to proportional hazards model, where Cox regression is by far the most prominent. Developments in FHT regression are relatively recent, and there has to our knowledge been no attempt at tackling it in the high-dimensional case, in which boosting is an appropriate choice of method.

High-dimensional survival analysis.

These days, genetic information is cheap. The cost of perform gene sequencing is very low, at the point where it is feasible to perform it on many patients.

The Cox regression model (Cox and Miller, 1965) is widely used to model survival data.

The main goal of this thesis is to adapt first-hitting-time models for survival analysis to a high-dimensional setting. This will be done by implementing a gradient boosting algorithm. Gradient boosting algorithms are iterative algorithms for performing penalized maximum likelihood estimation. Most traditional gradient boosting algorithms are concerned with modelling one parameter. We will estimate two parameters. To this end, we will review recent developments in gradient boosting for multidimensional likelihood functions.

In chapter 2, we will discuss first-hitting-time models.. We will first provide background on survival data and then briefly discuss Cox regression. The latter part of the chapter will be focused on first-hitting-time models. In chapter 3, we discuss gradient boosting methods. In chapter 4, we discuss a gradient boosting algorithm for first-hitting-time models. In chapter 5, we perform a simulation study. In chapter 6, we apply the developed algorithm on a real-life dataset with neuroblastoma patients, and compare it to Cox regression. Finally, chapter 7 summarizes and hints at future work.

Chapter 2

First-hitting-time models

2.1 Survival data and basic definitions

Survival analysis is the field of studying lifetime and time-to-event data. It has very important applciations in biomedical statistics, where we are often interested in, for example, the factors contributing to why some cancer patients recover, and others experience recurrence. An overview of modelling survival data is Aalen et al. (2008).

Figure out where the citation should be put.

2.1.1 General idea

Consider studying the time to an event of interest. You might 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. While these events are a relapse and a birth, respectively, we will collectively refer to them as events. The time-to-event T is a stochastic variable and of a non-negative domain. 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 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 τ. Luckily, not every patient relapses during that time, and so the actual time to their relapse, $\tilde{T}_i$, is not observed. Note that it might turn out that the actual $\tilde{T}_i$ does not exist. We could throw away these observations without an observed event. 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* data.

2.1.2 Independently censored survival data

subsec:survdata

An observed lifetime T is censored if the actual lifetime $\tilde{T}$ is larger than T. We can say that we have a censoring mechanism which works such that the observed T is $T = \min(\tilde{T}, C)$, where C is a certain censoring time. In the clinical trial example mentioned, the censoring time C would be the end of the

possible observation period of the trial, namely τ. To go with the censoring mechanism, we need a censoring indicator, which we denote d. It is defined as

$$d = \mathrm{I}(T = \tilde{T}), \tag{2.1}$$

{eq:censoring}

indicating if we have observed the actual event. If the event has occurred, the indicator d is 1, and if not, it is 0. An important definition 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

$$P(t|d) = P(t).$$

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

In survival analysis, one of the things we are interested in is the survival function. The survival function $S(t)$ is the probability of surviving until time t,

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

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) $f(t)$.

We are also interested in 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.

Find and add citation.

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

$$\alpha(t) = \frac{1}{S(t)} \lim_{\epsilon \to 0} \frac{F(t + \epsilon) - F(t)}{\epsilon} = \frac{f(t)}{S(t)} = \frac{-S'(t)}{S(t)}, \tag{2.2}$$

{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

$$\alpha(t) = \frac{S'(t)}{S(t)} = \frac{\frac{\mathrm{d}S}{\mathrm{d}t}}{S(t)}. \tag{2.3}$$

{eq:alpha-diff}

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

$$A(t) = \int_0^t \alpha(s) \, \mathrm{d}s, \tag{2.4}$$

{eq:cumulative-hazard-1}

and we insert (2.3) into (2.4),

$$A(t) = -\int_0^t \frac{\frac{\mathrm{d}S}{\mathrm{d}s}}{S(s)}\,\mathrm{d}s = -\int_0^t \frac{1}{S(s)}\,\mathrm{d}S = -\log(S(t)), \qquad (2.5)$$

<div style="float:right">{eq:cumulative-hazard}</div>

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,

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

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

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

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),

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

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

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

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,

$$D = (t_i, d_i, \boldsymbol{x}_i)_{i=1}^N. \qquad (2.6)$$

<div style="float:right">{eq:surv-D}</div>

Now, consider a survival time distribution

$$\psi(\boldsymbol{\beta}), \qquad (2.7)$$

<div style="float:right">{eq:surv-time-dist}</div>

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 (2.6), 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

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

{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

$$S(t_i|\boldsymbol{\beta}, \boldsymbol{x}_i) \tag{2.9}$$

{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 (2.8) and (2.9) 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

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{N} f(t_i|\boldsymbol{x}_i, \boldsymbol{\beta})^{d_i} S(t_i|\boldsymbol{x}_i, \boldsymbol{\beta})^{1-d_i}. \tag{2.10}$$

{eq:surv-lik}

It is more convenient to work with the log likelihood,

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

Note that since $A(t) = -\log S(t)$ and $f(t) = \alpha(t)S(t)$, see (2.5) and (2.2), respectively, (**??**) 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 $\psi(\boldsymbol{\beta})$ (2.7) for which either form of $l(\boldsymbol{\beta})$ can be calculated, we can perform maximum likelihood estimation of its parameters.

2.2.2 Proportional hazards regression

`subsec:ph-reg`

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. The reasons for this will become clear.

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

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

`{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 $\boldsymbol{\beta} = (\beta_1, \dots, \beta_p)$. We choose $r(\boldsymbol{x})$ such that it is appropriately normalized, meaning $r(\boldsymbol{0}) = 1$. 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, 1992).

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,

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

`{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 contributes to the likelihood with

$$\frac{\alpha_0(t_i) r(\boldsymbol{x}_i)}{\sum_{j \in R(t_i)} \alpha_0(t_i) r(\boldsymbol{x}_j)}. \qquad (2.13) \quad \boxed{\texttt{\{eq:hazard-frac\}}}$$

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

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

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, 1992). 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. Cox regression is used very much in applied research.

2.2.3   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.

2.2.4   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.

Find citation for this paragraph!

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.

2.3.1   General idea

fht-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

$$T = \inf_t \left(t \colon Y(t) \in \mathcal{B} \right). \qquad (2.15)$$

{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

$$P(T = \infty) = 1 - P(T < \infty). \qquad (2.16)$$

{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.

2.3.3 Wiener process as health 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,

$$Y(t) = y_0 + \mu t + \sigma W(t), \tag{2.17}$$

{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).

2.3.4 Wiener process and inverse Gaussian first hitting time

If we choose the stochastic process to be a Wiener process with intercept and drift (2.17), and we let the boundary be the non-positive numbers, $\mathcal{B} = (-\infty, 0]$, then (2.15) 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)

$$f(t|y_0, \mu, \sigma^2) = \frac{y_0}{\sqrt{2\pi\sigma^2 t^3}} \exp\left[-\frac{(y_0 + \mu t)^2}{2\sigma^2 t}\right], \tag{2.18}$$

{eq:ig-pdf}

and cumulative distribution function (cdf)

$$F(t|\mu, \sigma^2, y_0) = \Phi\left[-\frac{\mu t + y_0}{\sqrt{\sigma^2 t}}\right] + \exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left[\frac{\mu t - y_0}{\sqrt{\sigma^2 t}}\right]. \tag{2.19}$$

{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 (2.18) is improper, and the probability of the time not being finite is

$$\Pr\left(T = \infty\right) = 1 - \Pr\left(T < \infty\right) = 1 - \exp\left(-2 \cdot y_0 \cdot \mu\right), \qquad (2.20) \qquad \boxed{\texttt{\{eq:P-inf-FHT\}}}$$

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

$$S(t|\mu, \sigma^2, y_0) = \Phi\left[\frac{\mu t + y_0}{\sqrt{\sigma^2 t}}\right] - \exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left[\frac{\mu t - y_0}{\sqrt{\sigma^2 t}}\right], \qquad (2.21) \qquad \boxed{\texttt{\{eq:ig-surv\}}}$$

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 (2.21) 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

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

2.3.5   Effects of parameters in the Wiener process

Assume now that the process will reach 0 with certainty, such that $P(T = \infty)$ (2.16) is 0. Again, this will happen if the drift is non-positive. Let us consider a case where μ is strictly negative, and not zero. 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 nearly a deterministic function of y_0 and μ, such that

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

This is quite visible in Figure 2.3.5, which shows examples of 5 Wiener process paths with diffusion parameter $\sigma^2 = 1$, initial value $y_0 = 10$, and drift $\mu = -1$. 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).

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.

Figure 2.1: Example of 5 Wiener process paths with initial value $y_0 = 10$ and drift $\mu = -1$.
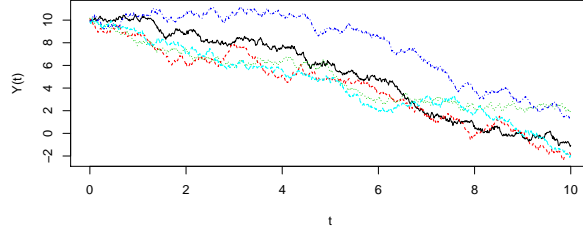
2.4 Properties of the inverse Gaussian

2.4.1 Inverse Gaussian distribution and different parameterizations

The typical inverse Gaussian parameterization is.

> Add this parameterization!

We will use ... This parameterization is not in the exponential family, and hence it is not a generalized linear model.

2.4.2 The inverse gaussian is overdetermined if the health process is latent

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 (2.18) and the survival function $S(t|\mu, \sigma^2, y_0)$ in (2.21) 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 the convention.

2.4.3 The shape of the hazard function in the inverse Gaussian model

From (2.2), 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. If y_0 is close to zero, we essentially get a decreasing hazard rate. If y_0 is far from zero, however, we essentially get an increasing hazard rate. If y_0 is somewhat inbetween, we get a hazard rate which first increases and then decreases (Aalen et al., 2008). In all three cases, the hazard has a peak. This peak is dependent on both the initial level and the drift, of course. It is not as simple as the peak being at the

mean lifetime, i.e., y_0/μ. Although the height of the peak will change both by changing both parameters, to simplify it a lot, y_0 mostly impacts at what time the peak is, whereas μ mostly affects the height of the peak. Regardless of the initial value, the hazard rate converges to the same limiting hazard, which is

$$\lim_{t\to\infty} \alpha(t) = \frac{1}{2}\left(\frac{\mu}{\sigma}\right)^2 = 0.5\mu^2,$$

as seen in Aalen et al. (2008). To get an intuitive feel for this, I have made an interactive website at `vegarsti.shinyapps.io/FHT_Hazard`

Add plots and more explanation here.

2.4.4 Comparison of hazard rates

It might be of particular interest to look at the ratio between two hazard rates. We might for example look at it when the drift μ is the same, but the initial level y_0 is different. Then the hazard ratio is strongly decreasing. This feature is the same phenomenon as that observed in frailty models, where the relative hazards often decline (Aalen et al., 2008).

Add plot and explain better.

It is also of interest to do the converse, that is, look at the hazard ratio when the initial level is the same, but the drift is different. The result here is quite different. The ratio of the hazards has a "bathtub" shape, which levels off at a later time (Aalen et al., 2008). Keep in mind here that levelling off means getting to proportional hazards. 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.

Add plot here as well; also here see ABG page 402

2.5 Regression with first-hitting-time models

subsec:IG-reg

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 model (Lee and Whitmore, 2006; Caroni, 2017) is to simply use the identity link function for the drift μ, and to use the logarithm link function for the initial level y_0, since in our framework it must be positive:

$$\mu(\boldsymbol{\beta}) = \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x} = \sum_{j=1}^{p}\beta_j x_j, \qquad (2.22) \qquad \text{\{eq:y0\}}$$

$$y_0(\boldsymbol{\gamma}) = \exp(\boldsymbol{\gamma}^{\mathrm{T}}\boldsymbol{z}) \Rightarrow \ln y_0(\boldsymbol{\gamma}) = \boldsymbol{\gamma}^{\mathrm{T}}\boldsymbol{z} = \sum_{j=1}^{d}\gamma_j z_j. \qquad (2.23) \qquad \text{\{eq:mu\}}$$

Here $\boldsymbol{\beta} \in \mathbb{R}^p$ and $\boldsymbol{\gamma} \in \mathbb{R}^d$ are vectors of regression coefficients. Note that we may let $\boldsymbol{x}$ and $\boldsymbol{z}$ share none, some, or all elements.

Plugging in the pdf (2.18) and the survival function (2.21) into the log-likelihood (2.10), we get that the log-likelihood of a survival data set with the

inverse gaussian FHT model, is

$$l(y_0, \mu, \sigma) = \sum_{i=1}^{n} d_i \left(\ln y_0 - \frac{1}{2} \ln\left(2\pi\sigma^2 t_i^3\right) - \frac{(y_0 + \mu t_i)^2}{2\sigma^2 t_i} \right)$$
$$+ (1 - d_i) \ln\left(\Phi\left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}} \right) - \exp\left(-\frac{2y_0\mu}{\sigma^2} \right) \Phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}} \right) \right).$$
$$(2.24)$$

{eq:loglik}

2.5.1 Fitting an IG FHT model

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 doing this exist.For R (R Core Team, 2013), the `threg` package (Xiao et al., 2015). There does not exist any method to fit either a *regularized* model at the moment, nor to do automatic variable selection.

2.5.2 Constructing survival probabilities based on estimates

Since we have a parametric expression for the survival function $S(t)$ (2.21), we can construct parametric estimates of the survival probability at time t, by plugging in estimates the parameters, $\hat{y}_0$ and $\hat{\mu}$. Since σ^2 is 1, we remove it from the expression as well. We denote the estimated probability for $\hat{S}(t)$,

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

2.5.3 Combining clinical and genetic data in the Inverse Gaussian FHT model

subsec:FHT-combine

Combining clinical and genetic data in biomedical survival analysis is a large and important goal. The IG FHT model lends itself nicely to combining clinical and genetic data. As discussed in Aalen and Gjessing (2001), an a priori decision of splitting covariates as those relating to "fixed" covariates, e.g. genes, and covariates related to "lifestyle," would be an instance of this combining of data. With this perspective, it would be reasonable to let the initial health level y_0 be a function of the "fixed" covariates, while the drift would be a function of the "lifestyle" ones. Furthermore, since genes are high-dimensional, this yields an easy way to combine high-dimensional data and low-dimensional data in the same model. Other approaches used, e.g. related to Cox regression, require tuning and weighting of parameters.

more here

Chapter 3

Statistical 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: Could any weak learner be transformed to become a strong learner? (Kearns and Valiant, 1989) 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, then, 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), which 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 $(\boldsymbol{x}_i, y_i)_{i=1,\dots,N}$, where $\boldsymbol{x}_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$, 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 $\boldsymbol{x}_{\text{new}}$ of the same form. Some observations are hard to classify, whereas some are not. One way to look at binary classification is to imagine the p-dimensional space of the observations $\boldsymbol{x}$, and think of the classifier as finding the line which best splits the observations into their corresponding label. Some observations are not at all close to the boundary, and so they

are easily classified. The problems start when the observations are close to
the boundary. Freund and Schapire (1996) realized that one could assign a
weight to each observation. First, assign equal weight to each observation.
Then, use a weak learner $h(\cdot)$ to make an initial classifier, minimizing the
weighted sum of misclassified points. Finally, based on the misclassification
rate of this classifier, calculate a weight α to give to this classifier. Currently,
the classifier is $F_1(\cdot) = \alpha_1 h_1(\cdot)$. After this initial classification, some points will
be correctly classified, and some will be misclassified. We increase the weights
of the misclassified ones, and normalize the weights afterwards. This then also
results in the correctly classified ones having a reduced weight. In the next
iteration, apply again a weak learner which minimizes the weighted sum of the
observations and reweight observations accordingly as before. Again, calculate
a weight to give to this new classifier, and add it to the previous classifier,
such that $F_2(\cdot) = \alpha_1 h_1(\cdot) + \alpha_2 h_2(\cdot)$. Continue iterating in this fashion until
an iteration m. The resulting final classifier, the AdaBoost classifier, becomes
$\hat{F}(\cdot) = F_m(\cdot) = \sum_{i=1}^{m} \alpha_i h_i(\cdot)$. It is a linear combination of the weak classifiers,
and 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: 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 also is 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 (overfitting 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.
We therefore extend our discussion to a more general statistical regression
scheme. 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.

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

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

containing the values of an outcome variable $\boldsymbol{y}$ and predictor variables $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_p$,
forming covariate matrix $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_p)$. We assume that the samples
$i = 1, \dots, n$ 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 is 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 θ as a functional of the covariates $\boldsymbol{X}$, with conditional expectation given the observed value $\boldsymbol{x}$ as

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

where $g(\cdot)$ is a so-called link function and $f(\cdot)$ is an additive predictor. We see that if we use $g^{-1}(\cdot)$, the inverse of the link function, on this expression, we get

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

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

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

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 the additive effects of the single predictors. This is called a GAM, and is specified by

$$f(\boldsymbol{x}) = \beta_0 + f_1(x_1) + \ldots + f_p(x_p), \tag{3.3}$$

{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 the partial effects of the variables x_j. The generic notation $f_j(x_j)$ may be 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.In statistical boosting algorithms, we typically use component-wise effects, meaning that the different partial effects are estimated by separate base-learners $h_1(\cdot), \ldots, h_p(\cdot)$. Read more about this in section 3.6 on component-wise boosting. 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.

We evaluate the fit of the model and its additive predictor using a loss function $\rho(y, f(\cdot))$, which 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 reveals, we wish to minimize this. We can 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 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.

3.2.1 Example of a model and corresponding loss function

Let us look at a specific example of a setting, using the notation described above. We have a dataset $D = \{\boldsymbol{x}_i, y_i\}_{i=1}^{N}$ where the responses y_i are continuous, and which we assume follow a normal distribution given the data. Thus we wish to model the conditional mean μ, and so we use μ in place 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 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

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

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. With all those parts in place, we can model the additive predictor $f(\cdot)$.

3.2.2 Model selection and model assessment

Test error, also referred to as *generalization error*, is the prediction error over an independent test sample

$$\text{Err}_\tau = \mathbb{E}[L(Y, \hat{f}(X)) | D] \tag{3.4}$$

where both X and Y are drawn randomly from their joint distribution (population). Here the training set D is fixed, and test error refers to the error for this specific training set. A related quantity is the expected prediction error (or expected test error)

$$\text{Err} = \mathbb{E}[L(Y, \hat{f}(X))] = \mathbb{E}[\text{Err}_D]$$

Note that this expectation averages over everything that is random, including the randomness in the training set that produced $\hat{f}$.

Estimation of Err_D will be our goal, although we will see that Err is more amenable to statistical analysis, and most methods effectively estimate it.

Training error, sometimes called empirical risk, is the average loss over the training sample

$$R = \overline{\mathrm{err}} = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}(x_i)).$$

We would like to know the *expected* test error of our estimated model $\hat{f}$,

$$\mathbb{E}[R] = \mathbb{E}[\overline{\mathrm{err}}] = \mathbb{E}\left[\frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}(x_i))\right].$$

As the model becomes more and more complex, it uses the training data more and is able to adapt to more complicated underlying structures. Hence there is a decrease in bias but an increase in variance. The bias here refers to assumptions made on the structure of the response, which is not based on the data. An example of a model with high bias is a linear model

$$y = \beta_0 + \sum_{j=1}^{p} \beta_j x_j.$$

It has high bias because it is only able to model a response y which is linear in each covariate x_j. A model with high variance, on the other hand, does not make many assumptions on the relationship. Given two observations $\boldsymbol{x}_1$ and x_2, which are relatively similar, a model with high variance may propose two responses which are relatively different from each other. At least more so than a model with high bias, but low variance. There exists some intermediate model complexity that gives minimum expected test error, which is what we want to achieve.

Unfortunately training error is not a good estimate of the test error. Typically, the training error will drop to zero if we increase the model complexity enough. This is because, as the bias becomes very small and the variance very large, the model ends up effectively learning the randomness inherent in the data, and not the structure of the relationship between the input X and the output Y. This is what is called overfitting. An overfit model generalizes poorly, and it is generalization that we are after.

So how do we estimate the expected test error of a model $\hat{f}$? Typically our model will have a tuning parameter α, and so we write $\hat{f}_\alpha$. This tuning parameter varies the complexity of our model, and we wish to find that value of α which minimizes error, i.e., which produces the minimum of the average test error.

It is important to note that there are two separate goals that we have in mind. First, we wish to perform **model selection**. We wish to choose the best model, and to do this we need to estimate the performance of different models. This is typically done in two steps. First, the model $\hat{f}_\alpha$ needs to be estimated. This is done using what is called a *training set*. Second, we estimate the prediction error of this model on a *validation set*. Having done so, for various values of α, we wish to perform **model assessment**, to find the α

which minimizes generalization error. We wish to, as accurately as possible, estimate the prediction error (generalization error) of our chosen model. This is typically done using a so-called *test set*. It is important that we do not consider the performance of our chosen model on the validation set as the prediction error. This estimate is biased, because we are only looking at the performance of this model *because* its performance is good. Ideally, the test set should be kept completely separate, and be brought out only at the end of the data analysis.

In a setting where data is abundant, one can simply split the data into these three parts: Training, validation, and test. However, often, data is not abundant, and we wish to use the data as efficiently as possible. There exist good methods to do this. Notable general methods are the bootstrap, and K-fold cross-validation. We will in this thesis not discuss the bootstrap, but we will discuss cross-validation in- depth later, in subsection 3.8.1. In short, however, K-fold cross validation uses a training data set and uses this as a joint training and validation set. First, split the data set into a test set and an original training set. Then, split the original training set into K equally sized parts. For each part, *train/estimate* the model $\hat{f}_\alpha$ on $K - 1$ of the parts, and *validate* on the K-th (the last) part. Then do this K times, treating each part in turn as a validation set. Then sum up the validation errors across the K validation sets. This means that the entire, original training set has also been used as a validation set. There is still a bias, but it is smaller than validating on the same data as training.

Our goal will be to use gradient boosting to minimize 3.4. 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 $G\colon \mathbb{R}^m \to \mathbb{R}$, where $m \in \mathbb{N}$. Gradient descent is a 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 $\boldsymbol{x} = (x_1, x_2, \ldots, x_m)$ exist, then the gradient of G at $\boldsymbol{x}$ is the vector of all its partial derivatives at $\boldsymbol{x}$, namely

$$\nabla G(\boldsymbol{x}) = \left(\frac{\partial G(\boldsymbol{x})}{\partial x_1}, \frac{\partial G(\boldsymbol{x})}{\partial x_2}, \ldots, \frac{\partial G(\boldsymbol{x})}{\partial x_m} \right).$$

The motivation behind the gradient descent algorithm is that in a small interval around a point $\boldsymbol{x}_0 \in \mathbb{R}^m$, G is decreasing the most in the direction of the negative gradient at that point. Therefore, by taking a small step slightly in the direction of the negative gradient, 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 less than $G(\boldsymbol{x}^{[0]})$. In some versions of the algorithm, the step length $\nu \in (0, 1]$ is found by a line search, i.e., by finding the step length which gives the best improvement. In other versions, one simply uses a fixed step length. The gradient descent algorithm repeats this procedure until convergence. Indeed, with a sufficiently small step length, 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 often seems to find good solutions globally. This is likely related to research which has found that in high-dimensional spaces,

---

Algorithm 1 Gradient descent

We want to minimize $G(\boldsymbol{x})$, i.e. solve $\min_{\boldsymbol{x}} G(\boldsymbol{x})$.

1. Start with an initial guess $\boldsymbol{x}^{[0]}$, for example $\boldsymbol{x}^{[0]} = \boldsymbol{0}$, and set m to 0.

2. Increase m by 1.

3. Calculate the direction to step in, $\boldsymbol{g}_{m-1} = -\nabla G(\boldsymbol{x}^{[m-1]})$.

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. If $m < m_{\text{stop}}$, go to step (2).

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

---

most minima are not minima, but in fact, saddlepoints masquerading as local minima (Dauphin et al., 2014). This means that training will slow since the gradient will be small at this saddlepoint or plateau. When using a gradient descent method typically one sets a threshold at which the algorithm terminates when the gradient becomes smaller than the threshold. However if powering through the saddlepoint, then the multivariate gradient descent search should be able to continue digging downwards from these points.

3.4 The gradient boosting approach

3.4.1 Gradient boosting

In a seminal paper, Friedman (2001) developed an iterative algorithm for fitting an additive predictor (3.3). He called the algorithm gradient boosting. He showed that AdaBoost performs this algorithm for a particular loss function, namely the exponential loss function. This provided a connection between what had previously been in the machine learning domain, with the statistical domain. See Hastie et al. (2009) for a good demonstration of this argument. Friedman's work provided a way of viewing boosting through a statistical lens, and connected the successful machine learning approach to the world of statistical modelling. Gradient boosting can be viewed as an optimization procedure in functional space, because gradient boosting is gradient descent in the functional parameter space spanned by the base learners (Friedman, 2001; Bühlmann and Hothorn, 2007).

Consider the task of deriving a general prediction $\hat{\eta}$ by minimizing the expectation of a loss function $\rho(\cdot, \cdot)$ assumed to be differentiable with respect to η:

$$\hat{\eta} = \operatorname*{argmin}_{\eta} \left(\mathbb{E}_{Y,X}[\rho\{Y, \eta(X)\}] \right),$$

where Y and X are the random variables for response and covariates, respectively. η here denotes a general prediction. In practice, to estimate a predictor based on a sample of observations, we minimize the empirical risk R_D of the data set $D = (\boldsymbol{x}_i, y_i)_{i=1}^n$,

$$R_D(\eta) = \frac{1}{n} \sum_{i=1}^n \rho(y_i, \eta(\boldsymbol{x}_i)).$$

To avoid overfitting, we impose sufficient regularization. This is easy to do in a gradient boosting algorithm.

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

Now, consider the problem of finding a predictor which minimizes the empirical risk of a chosen loss function on a data set $D = \{x_i, y_i\}_{i=1}^N$,

$$\eta^* = \operatorname*{argmin}_{\eta} R_D(\eta) = \operatorname*{argmin}_{\eta} \sum_{i=1}^n \rho(\boldsymbol{y}_i, \eta(\boldsymbol{x}_i)), \qquad (3.5)$$

{eq:argmin-eta}

where the parameter η is a predictor. The gradient boosting algorithm is a way to build up such a predictor η by way of iterative fitting of base learners h.

The approach proposed by Friedman (2001) is to take inspiration from gradient descent and let $\hat{\eta}$ be a sum

$$\hat{\eta} = \beta_0 + \sum_{m=1}^{m_{\text{stop}}} f^{[m]},$$

where the first term, β_0, is an initial guess, usually $\overline{y}$, and the remaining set $\{\hat{f}^{[m]}\}_{m=1}^M$ of functions is a set of increments – steps, or boosts – defined by the optimization method. Note that this structure is the same as the decomposition of the solution to the gradient descent algorithm.

To perform gradient descent on our objective function, the empirical risk $R_D(\cdot)$, we need to compute the negative gradient of the loss function, with respect to the additive predictor. For a response y dependent on covariates $\boldsymbol{x}$, this is

$$-\frac{\partial}{\partial \eta} \rho(y, \eta(x)),$$

We denote the realization of the negative gradient on the observed data by $\boldsymbol{u}$, and call $\boldsymbol{u}$ *generalized residuals*, and they are

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

where $\hat{\eta}^{[m-1]}$ is the estimate of the additive predictor at the previous iteration. We now have a vector of generalized residuals, where each element i is a measure of the error that the model in iteration $m-1$ makes in trying to predict the outcome y_i given $\boldsymbol{x}_i$. With the generalized residuals in hand, we should now be able to perform a gradient descent step, which should lead us closer to a solution to the minimization problem. We can treat each $\eta(x_i)$ as a parameter to optimize. By using the gradient descent algorithm directly, we can then calculate the optimal step to take, and add an increment to the estimate $\hat{\eta}(x_i)$, for each $i = 1, 2, \ldots, n$.

3.4.2 Functional Gradient Boosting

However, while this nonparametric approach would reduce the error of each data point, it will not generalize to a similar data set where one observes different data points. 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 in XXXX, over all values of $\boldsymbol{X}$ and Y in the joint distribution. Additionally, we wish to have an interpretable model.

Find correct equation to reference for minimization

Therefore, 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*. 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. Each parameterized step is a realization of a so-called base learner

$$h(\cdot).$$

A base learner is usually a relatively simple parametric effect of β. Again, typical examples are linear least squares, stumps (trees with one split; see Bühlmann and Hothorn (2007) and Hastie et al. (2009)), and splines with a few degrees of freedom. There are several reasons to use simple base learners. One is that there often exists fast methods for estimating a single base learner. Therefore there will be little computational cost in each step. econdly, there is relatively more to gain by combining simple learners, rather than combining complex learners.

For the functional gradient descent, we still start with an initial value for η, a constant β_0. Then we iterate, let us say, at each step $m > 0$ first calculating the generalized residuals of the previous iteration,

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

like we have seen before. Here we insert the model from the previous step, $\hat{\eta}^{[m-1]}$. We now perform a gradient descent step, but we are constrained to steps which are functions of the base learner $h(\cdot)$. In a functional sense, to take the steepest gradient, we must choose the relization of the base learner which produces the function $\hat{h}^{[m]}$ *most parallel* to $\boldsymbol{u}^{[m-1]}$. Another way of looking at it is that it this is the h most correlated with $\boldsymbol{u}^{[m-1]}$ over the data distribution. This means that this $\hat{h}^{[m]}$ is an approximation of the generalized residuals $\boldsymbol{u}^{[m-1]}$, or, a projection of the generalized residuals onto the space spanned by the base learner function class. We obtain that $\hat{h}_m$ by fitting the base learner $h(\cdot)$ to the generalized residuals. The method of fitting depends on the base learner. If, for example, the base learner is a linear regressor, then this will be $\hat{h}^{[m]} = \left(\boldsymbol{u}^{[m-1]} \right)^{T} \hat{\boldsymbol{\beta}}^{[m]}$, where

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

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,

$$a^{[m]} = \operatorname*{argmin}_{a} R_D \left(\hat{f}^{[m-1]} + a^{[m]} \cdot \hat{h}^{[m]} \cdot \right).$$

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

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

We iterate this procedure until some stopping criterion. The resulting model

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

has an additive structure which 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, see Algorithm 2.

3.4.3   Tuning parameters

3.4.3.1   Step length

In the original generic functional gradient descent algorithm, the step length a_m for each iteration is found by a line search. Friedman (2001) says that fitting the data too closely may be counterproductive, and result in overfitting. To combat the overfitting, one constrains the fitting procedure. This constraint is called regularization. Friedman therefore, later in the paper, proposes to regularize each step in the algorithm by a common learning rate, $0 < \nu \leq 1$. Another natural way to regularize would have been to control the number of terms in the expansion, i.e., number of iterations, M. However, it has often been found that regularization through shrinkage provides superior results (Copas, 1983).

As we will see, most modern boosting algorithms omit the step of the line search to find a_m, but instead uses a fixed learning rate/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., produces sufficient shrinkage, but the convention is to use $\nu = 0.1$ (Mayr et al., 2014a). This reduces the complexity of the algorithm, and reduces the number of tuning parameters to 1, the number of steps, or iterations. There will of course be a tradeoff between the number of iterations M and the size of the step length ν, which is another reason to use the conventional step length each time.

3.4.4   Number of iterations

subsec:
iterations

With a fixed step length (learning rate), the main tuning parameter for gradient boosting is the number of iterations M that are performed before the algorithm is stopped. Its value is critical: If M 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.

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 $\rho(y, \hat{f}(x))$, for which we wish to minimize the empirical risk, i.e., the loss function evaluated on the samples,

$$\hat{f} = \operatorname*{argmin}_f R(f) = \operatorname*{argmin}_f \sum_{i=1}^n \rho(y_i, f(x_i)).$$

2. Set iteration counter m to 0. Initialize the additive predictor by setting $\hat{f}_0(\cdot)$ to a constant β_0. One option is to find the the best constant my numerical maximization, i.e.,

$$\beta_0(\cdot) = \operatorname*{argmin}_c R(c).$$

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

4. Increase m by 1.

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

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

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

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

$$a^{[m]} = \operatorname*{argmin}_a R\left(\hat{f}^{[m-1]} + a \cdot \hat{h}^{[m]} \cdot \right).$$

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

9. Repeat steps 4 to 8 (inclusive) until $m = m_{\text{stop}}$.

10. Return $\hat{f}(\cdot) = \hat{f}^{[m_{\text{stop}}]}(\cdot) = \sum_{m=0}^{m_{\text{stop}}} f^{[m]}(\cdot)$.

3.5 L_2**Boost**

With the generic functional gradient boosting algorithm seen in section 3.4.2, it is 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.

In the original paper (Friedman, 2001), Friedman derived an algorithm for the standard regression setting, which he called L_2Boost. L_2Boost is a computationally simple variant of boosting, constructed from a functional gradient descent algorithm for the L_2 loss function,

$$\rho(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2.$$

The reason it is simple is that the generalized residual u_i of an observation y_i, x_i, i.e., the negative derivative of the loss function with regard to an estimate $\hat{y}_i = \hat{f}(x_i)$, is

$$-\frac{\partial}{\partial \hat{y}}\rho(y_i, \hat{y}_i) = y_i - \hat{y}_i,$$

that is, the so-called residual. The negative gradient vector $\boldsymbol{u}$ then becomes simply the residual vector,

$$\boldsymbol{u} = \left(\frac{\partial \mathrm{L}(y, f(\boldsymbol{x}))}{\partial x_i}\right)_{i=1}^{n} = (y - f(x_i))_{i=1}^{n},$$

and hence the algorithm results in repeated refitting of residuals (Friedman, 2001; Bühlmann and Yu, 2003). With $M = 2$ iterations, this had in fact been proposed already, under the name of "twicing" (Tukey, 1977). See Algorithm 3 for an overview of the algorithm. Note that we here use the algorithm given in Bühlmann and Yu (2003), who do not use a step length, i.e., they let $\nu_m = \nu = 0.1$ for all iterations $m = 1, \ldots, M$.

3.6   **High dimensions and component-wise gradient boosting**

sec:component

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 ($p >> N$), 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", which means that shrinkage of coefficient estimates or selection of relevant predictors is carried out simultaneously with the estimation of the model parameters. Both shrinkage and variable selection will typically improve prediction accuracy: In case of shrinkage, coefficient estimates tend to have a slightly increased bias but a decreased variance, while in case of variable selection, overfitting the data is avoided by selecting only the most informative predictors. For instance in the L_2Boost algorithm, if one uses a least squares base learner which uses all p dimensions, we see that it is infeasible: The matrix which must be inverted is singular when the number of

Algorithm 3 L_2Boost

1. Start with a data set $D = \{x_i, y_i\}_{i=1}^N$. Set the loss function $\rho(y, \hat{f}(x)) = \frac{1}{2}(y - \hat{f}(x))^2$, for which we wish to minimize the empirical risk, i.e., the loss function evaluated on the samples,

$$\hat{f} = \operatorname*{argmin}_f R(f) = \operatorname*{argmin}_f \sum_{i=1}^n y_i - \hat{f}(x_i).$$

2. Set $m = 0$. Initialize $f_0(\boldsymbol{x})$, e.g., by setting it to zero for all components, or by finding the best constant, i.e.,

$$f_0(\cdot) = \operatorname*{argmin}_c R(c).$$

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

4. Increase m by 1.

5. Compute the negative gradient vector, i.e., the residuals, with the model evaluated at the previous estimate

$$\boldsymbol{u}^{[m-1]} = \left(y_i - \hat{f}(x_i)\right)_{i=1}^N$$

6. Estimate $\hat{h}_m$ by fitting $(\boldsymbol{x}_i, u_i^{[m-1]})$ using the base learner h (like in the previous algorithm):

$$\boldsymbol{\beta}_m = \operatorname*{argmin}_{\boldsymbol{\beta}} \sum_{i=1}^N \mathrm{L}(u_i^{[m-1]}, h(\boldsymbol{x}_i; \boldsymbol{\beta}))$$

This estimation can be viewed as an approximation of the negative gradient vector, and as the projection of the negative gradient vector onto the space spanned by the base learner.

7. Update $f_m(\cdot) = f_{m-1}(\cdot) + h(\cdot; \boldsymbol{\beta}_m)$.

8. Repeat steps 4 to 7 (inclusive) until $m = M$.

9. Return $\hat{f}(\cdot) = \hat{f}_{m_{\mathrm{stop}}}(\cdot) = \sum_{m=0}^{m_{\mathrm{stop}}} f_m(\cdot)$.

predictors p is larger than the number of observations N. For other models, it might be possible to estimate parameters for each predictor, but it would very easily result in overfitting. If, for instance, the data set input $\boldsymbol{X}$ consists of gene expressions, it is obvious that the response variable y is not dependent on every single gene.

3.6.1 Stagewise, not stepwise

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 the L_2Boost paper (Bühlmann and Yu, 2003), and is very much an active field of research (Bühlmann, 2006; Mayr et al., 2014a,b, 2017). In the gradient boosting algorithm described in algorithm (2), we start out with an additive predictor $f^{[0]}(\cdot)$ which only consists of a constant. We have not added any effects of covariates yet. Instead of adding a small effect from all predictors, the component-wise approach adds only the effect of one variable at a time. This is similar to the typical statistical model selection regime of forward stepwise model selection. In forward stepwise, we will iterate in the following manner. We start with an empty set of predictors, or covariates, and look at each separately. Looking at each separately means adding it to the set of predictors and estimating a model with those predictors. Then we add, to the set of predictors, that predictor which gives the best improvement to the objective function. We repeat this in each step, but we estimate the entire model in each step. The main idea of component-wise gradient boosting is to do this, except in a stagewise manner. This means that we do not change the added parameters, but we only estimate the next one.

Now, consider the problem of finding a predictor on the GAM form (3.3), which minimizes the empirical risk of a chosen loss function on a data set $D = \{x_i, y_i\}_{i=1}^N$,

$$\eta^* = \operatorname*{argmin}_{\eta} R_D(\eta) = \operatorname*{argmin}_{\eta} \sum_{i=1}^{n} \rho(\boldsymbol{y}_i, \eta(\boldsymbol{x}_i)), \tag{3.6}$$

`{eq:argmin-eta}`

where the parameter η is an additive predictor

$$\eta(\boldsymbol{x}) = g(\theta) = \beta_0 + f(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{p} f_j(x_j),$$

and $g(\cdot)$ is a known, monotonic link function appropriately chosen. Since we have component-wise effects, the model is interpretable and yields itself nicely to statistical inference, and, to a component-wise expansion of the gradient 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. Instead of using one 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. These base learners may have the same structure, or they may not, but importantly, they only use a single

covariate. For example, if we use a linear least squares model as base learners, the set of base learners would be

$$\mathcal{H} = \{h_1(\boldsymbol{x}; \beta_1) = \beta_1 x_1, h_2(\boldsymbol{x}; \beta_2) = \beta_2 x_2, \ldots, h_p(\boldsymbol{x}; \beta_p) = \beta_p x_p\}.$$

The initialization of the algorithm is the same as in the FGD algorithm: We first initialize the additive predictor to a constant β_0. In a given iteration m, we first derive the generalized residuals by calculating the negative gradient where we insert the additive predictor from the previous step,

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

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 now estimate all base learners separately, obtaining 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, and as the projection of the negative gradient vector onto the space spanned by the component-wise base learner. However, this time, they are projections onto only one component of the covariate space. To select the best-fitting base-learner $h_{j^{[m]}}^{[m]}$, 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 to the current model, with a pre-specified step length of ν. Hence the model after iteration m is

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

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

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

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 tuned stopping iteration m_{stop}. The final additive predictor becomes

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

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, i.e.,

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

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 4.

3.7   Boosting performs data-driven 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 columns of $\boldsymbol{x}$ 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.

3.8   Selecting m_{stop}

As we mentioned in subsection 3.4.4, the crucial tuning parameter in boosting is the number of iterations, m_{stop}. Stopping early enough performs 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, after all interested in minimizing out-of-sample prediction error (PE). The prediction error for a given data set is a function of the boosting iteration m. What we want is therefore a good method for approximating PE(m). This can be done in a number of ways. Many authors state that the algorithm should be stopped early, but do not go further into the details here. Common model selection criteria such as the Akaike Information Criteria (AIC) may be used, however the AIC is dependant on estimates of the

---

Algorithm 4 Component-wise gradient boosting

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

$$\hat{f} = \underset{f}{\operatorname{argmin}} R(f).$$

2. Set iteration counter m to 0. Specify a step length ν. Initialize the additive predictor to an offset by setting $f_0(\cdot)$ to a constant β_0. One option is to find the the best constant by numerical maximization, i.e.,

$$\beta_0 = \underset{c}{\operatorname{argmin}} R(c).$$

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 $\boldsymbol{X}$.

4. Increase m by 1.

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

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

6. For each base learner $h_j \in \mathcal{H}, j = 1, \ldots, p$, estimate $\hat{h}_j^{[m]}$ by fitting $(\boldsymbol{X}_i, u_i)$ 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,

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

8. Update the current model with the best-fitting model from the current iteration
$$\hat{f}^{[m]}(\cdot) \leftarrow \hat{f}^{[m-1]}(\cdot) + \nu \cdot \hat{h}_{j^{[m]}}^{[m]}(\cdot).$$

9. Repeat steps 4 to 8 (inclusive) until $m = m_{\text{stop}}$.

10. Return the final boosted additive predictor

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

---

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). This is problematic for several reasons. For L_2Boost, Bühlmann and Hothorn (2007) suggest that $\mathrm{df}(m) = \mathrm{trace}(B_m)$ 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. Instead, cross-validation, a very common method for selection of tuning parameters in statistics, is what is used in almost all cases, both in practice and in research (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.

3.8.1   K-fold cross-validation

K-fold cross-validation (Lachenbruch and Mickey, 1968), or simply cross-validation, is a general method commonly used for selection of penalty or tuning parameters because it approximates the prediction error. 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 in estimating the model. We often say that the k-th fold is left out. 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 for the prediction 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

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

For each m, we calculate the estimate of the cross-validated prediction error $\mathrm{CV}(m)$. We choose m_{stop} to be the minimizer of this error,

$$m_{\mathrm{stop}} = \operatorname*{argmin}_{m} \mathrm{CV}(m).$$

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.

3.8.2   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).

3.8.3 Repeated cross-validation

The randomness inherent in the cross-validation splits has an effect on the resulting m_{stop}. This is true for boosting in general, but it is especially true for real-life survival data. In typical survival time data sets one typically has a small effective sample size (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. 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 simply 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,

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

As before, we choose m_{stop} to be the minimizer of this error,

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

In practice, to ensure we find the minimizing m, 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.

3.9 Multidimensional boosting: Cyclical component-wise

A limitation of all the classical boosting methods we have described earlier, 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. 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).

3.10   GAMLSSBoost Algorithm

A key feature of GAMLSS 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 hamd, 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). 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).

   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

$$f_{\text{dens}}(y_i | \boldsymbol{\theta}_i), \tag{3.7}$$

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_{θ_k} and depends additively on the covariates. θ_k is linked to a 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

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

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 (3.7). The log-likelihood contribution for one sample is

$$\log\{f_{\text{dens}}(\boldsymbol{\theta}|y)\},$$

where y is the response of the sample and $\boldsymbol{\theta}$ is a vector of the distribution parameters, which will be a functional which works on the covariate vector $\boldsymbol{x}$. Hence the total log-likelihood is

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

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 $k = 1, 2, \ldots, K$. 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. 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, using the estimates of the other distribution parameters as offset values. We cycle through all parameter dimensions in each boosting iteration.

In any iteration, the algorithm cycles through the different parameter dimensions k. 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, however, uses the component-wise base learner strategy, introduced in section 3.6. This allows for, as we saw, model fitting in high-dimensional contexts, and in general, to be often powerful.

What does Riccardo mean here? Check. Also add citation.

To use the gradient boosting approach for a multidimensional prediction function, we need to have existing partial derivatives of the loss function with regard to each predictor. Since we are doing a gradient descent step, we use the *negative* derivative of the estimate. Since we have a prediction function which uses a vector of distribution parameters, we must take the partial derivatives. These negative partial derivatives are

$$-\frac{\partial}{\partial \eta_k} \rho(y, \boldsymbol{\eta}) = \frac{\partial}{\partial \eta_k} \log(f_{\text{dens}}(y|\boldsymbol{\theta})),$$

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 $\boldsymbol{u}_k$, $k = 1, 2 \ldots, K$. In a given iteration $m > 0$ and for a distribution parameter θ_k, we construct a generalized residual by computing the negative derivative with regard to each additive predictor η_k, and inserting the current estimate $\hat{\boldsymbol{\eta}}^{[m-1]}$, evaluated at each observation $(x_i, y_i)_{i=1}^n$. This yields

$$\boldsymbol{u}_k^{[m-1]} = (u_{k,1}^{[m-1]}, u_{k,2}^{[m-1]}, \ldots, u_{k,N}^{[m-1]})$$

$$= \left(-\frac{\partial}{\partial \eta_k} \rho(y, \boldsymbol{\eta})\big|_{\boldsymbol{\eta} = \hat{\boldsymbol{\eta}}^{[m]}} \right)_{i=1}^N.$$

Like in other gradient boosting algorithms, we need base learners. In this algorithm, we specify component-wise base learners for each distribution parameter. In principle, these might be different, but for simplicity, 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, we have base learners

$$h_{1,1}, \ldots, h_{1,p_1}, h_{2,1}, \ldots, h_{2,p_2}, \ldots, h_{K,p_K}.$$

Based on the covariates and the residuals, 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, typically zero. Alternatively, one might do a joint optimization of the log-likelihood, finding the optimal constant c_k for each distribution parameter. Then, initialize the estimate of each predictor η_k as

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

for each $k = 1, 2, \ldots, K$.

In the k-th step of iteration m, i.e. after having cycled through to 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 $m - 1$: $\hat{\eta}_{k+1}^{[m-1]}$, $\hat{\eta}_{k+2}^{[m-1]}$, 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 residual 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

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

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,

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

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 looks as follows:

$$
\frac{\partial}{\theta_1} \rho \left(y, \eta_1^{[m-1]}, \eta_2^{[m-1]}, \eta_3^{[m-1]}, \ldots, \eta_{K-1}^{[m-1]}, \eta_K^{[m-1]} \right) \xrightarrow{\text{calculate}} \boldsymbol{u}_1^{[m-1]} \xrightarrow{\text{fit and update}} \hat{f}_1^{[m]}
$$

$$
\frac{\partial}{\eta_2} \rho \left(y, \hat{\eta}_1^{[m]}, \hat{\eta}_2^{[m-1]}, \hat{\eta}_3^{[m-1]}, \ldots, \hat{\eta}_{K-1}^{[m-1]}, \hat{\eta}_K^{[m-1]} \right) \xrightarrow{\text{calculate}} \boldsymbol{u}_2^{[m-1]} \xrightarrow{\text{fit and update}} \hat{f}_2^{[m]}
$$

$$
\frac{\partial}{\eta_3} \rho \left(y, \hat{\eta}_1^{[m]}, \hat{\eta}_2^{[m]}, \hat{\eta}_3^{[m-1]}, \ldots, \hat{\eta}_{K-1}^{[m-1]}, \hat{\eta}_K^{[m-1]} \right) \xrightarrow{\text{calculate}} \boldsymbol{u}_3^{[m-1]} \xrightarrow{\text{fit and update}} \hat{f}_3^{[m]}
$$

$$
\cdots
$$

$$
\frac{\partial}{\eta_{K-1}} \rho \left(y, \hat{\eta}_1^{[m]}, \hat{\eta}_2^{[m]}, \hat{\eta}_3^{[m]}, \ldots, \hat{\eta}_{K-1}^{[m-1]}, \hat{\eta}_K^{[m-1]} \right) \xrightarrow{\text{calculate}} \boldsymbol{u}_{K-1}^{[m-1]} \xrightarrow{\text{fit and update}} \hat{f}_{K-1}^{[m]}
$$

$$
\frac{\partial}{\eta_K} \rho \left(y, \hat{\eta}_1^{[m]}, \hat{\eta}_2^{[m]}, \hat{\eta}_3^{[m]}, \ldots, \hat{\eta}_{K-1}^{[m]}, \hat{\eta}_K^{[m-1]} \right) \xrightarrow{\text{calculate}} \boldsymbol{u}_K^{[m-1]} \xrightarrow{\text{fit and update}} \hat{f}_K^{[m]}
$$

For a schematic overview of this cyclical multidimensional boosting algorithm, see Algorithm 5. 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. The main tuning parameters in this 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, it should not run until convergence, but rather find estimates by cross-validation (Schmid et al., 2010). An issue with that, though, is that for proper tuning it requires a vector $\boldsymbol{m}_{\text{stop}}$ of stopping iterations, i.e., one stopping iteration for each prediction parameter. To properly tune these parameters, it is necessary to perform a multidimensional search, which is usually done by implementing the 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.

3.10.1   Grid search cross-validation in gradient boosting

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 $m_{\text{min},k}$ and $m_{\text{max},k}$, respectively. We then divide this one-dimensional search space into a finite grid with N_k points, such that we obtain

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

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 $K = 3$, and $m_{\text{min},k} = 1$ and $m_{\text{max},k} = 10$ for all k, and finally 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:

$$(m_{1,1}, m_{1,2}, m_{1,3})$$
$$(m_{1,1}, m_{1,2}, m_{2,3})$$
$$\ldots$$
$$(m_{1,1}, m_{1,2}, m_{10,3})$$
$$(m_{1,1}, m_{2,2}, m_{1,3})$$
$$\ldots$$
$$(m_{1,1}, m_{2,2}, m_{10,3})$$
$$\ldots$$
$$(m_{10,1}, m_{10,2}, m_{10,3}).$$

We want to find the best configuration $\boldsymbol{m}$, i.e., we want to find the optimum of the hyperplane $CV(\boldsymbol{m})$. Like in subsection 3.8.1, we must calculate the estimate of the cross-validated prediction error for each given configuration $\boldsymbol{m}$, obtaining the prediction error $CV(\boldsymbol{m})$. We choose $\boldsymbol{m}_{\text{stop}}$ to be the minimizer of this error,

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

Using boosting, we may obtain estimates of $\text{CV}(\boldsymbol{m})$ for all $\text{CV}(\boldsymbol{m})$ by fixing all but one of the parameters and perform a typical boosting run. If we fix all but one of the parameters in the vector $\boldsymbol{m} = (m_{i_1,1}, m_{i_2,2}, m_{i_3,3})$, where $m_{\text{min},k} \leq i_k \leq m_{\text{max},k}$ for all $k = 1, 2, 3$, say, we fix $m_{i_1,1}$ and $m_{i_2,2}$. This is due to the way boosting algorithms work, since for any given iteration M, we also automatically obtain all boosted estimates for all iterations less than M, if

---

Algorithm 5 Multidimensional cyclical component-wise gradient boosting

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

$$\hat{\boldsymbol{\eta}} = \underset{\boldsymbol{\eta}}{\operatorname{argmin}}\, R(\boldsymbol{\eta}) = \underset{\boldsymbol{\eta}}{\operatorname{argmin}} \sum_{i=1}^n \rho(y_i, \boldsymbol{\eta})\big|_{\boldsymbol{\eta}=\boldsymbol{\eta}(x_i)}.$$

2. Initialize iteration counter m to 0. Initialize additive predictors $\beta_{0,1}^{[0]}, \beta_{0,2}^{[0]}, \ldots, \beta_{0,K}^{[0]}$. This can be done e.g., by setting these to zero, or by finding the best constants by maximizing the (joint) likelihood of the data.

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.

4. Increase m by 1.

5. Set k to 0.

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 $\hat{\boldsymbol{\eta}}^{[m-1]}(x_i), i = 1, \ldots, N$, yielding the negative gradient vector

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

8. Fit the negative gradient vector to each of the p components of $\boldsymbol{x}$ separately, using each component's respective base learner. This yields p vectors of predicted values,

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

where each vector is an estimate of the negative gradient vector $\boldsymbol{u}_k^{[m-1]}$, or, again, a projection onto the space spanned by the component-wise learner.

9. Select the component of $\boldsymbol{x}$ which fits best $\boldsymbol{u}_k^{(m-1)}$ (according to RSS), or, rather, the best-fitting base learner,

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

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

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

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,

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

Then update the full model thus far.

11. Repeat steps 6 to 10 until $m = \max(m_{\text{stop},1}, \ldots, m_{\text{stop},K})$.

12. Return $\hat{\boldsymbol{\eta}}$.

---

we have saved the boosted parameters for each iteration. Consider again the example. We now let the first two parameters in the example be fixed for each boosting run. While the search grid consist of $N_1 \cdot N_2 \cdot N_3$ tuples, considering the first two parameters as fixed, we only need to do $N_1 \cdot N_2$ boosting runs, and in each run set the maximum number of possible iterations in the boosting algorithm for the third component to be $m_{\max,3}$. This means that we consider all configurations of the first two parameters in $\boldsymbol{m}$, i.e.,

$$(m_{1,1}, m_{1,2})$$
$$(m_{1,1}, m_{2,2})$$
$$\dots$$
$$(m_{1,1}, m_{10,2})$$
$$(m_{2,1}, m_{1,2})$$
$$\dots$$
$$(m_{2,1}, m_{10,2})$$
$$\dots$$
$$(m_{10,2}, m_{10,2}),$$

and do a boosting run for each such. Like in subsection 3.8.1, we choose

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

where

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

i.e., the cross-validated prediction error, as usual.

3.11   Noncyclical component-wise multidimensional boosting algorithm

In the cyclical algorithm seen previously in Algorithm 5, 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 3.10.1, the usually applied *grid search* for such parameters scales exponentially with the number of parameters K. This can quickly become very demanding computationally. Thomas et al. (2018) develop a new algorithm for fitting GAMLSS models, instead of the cyclical one used in *gamboostLSS*. In this new algorithm, which they call "noncyclical," only one scalar tuning parameter m_{stop} is needed because only one parameter is chosen in each boosting iteration. 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).

3.11.1   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. If we want to avoid

having a separate tuning parameter for each parameter that we are boosting, however, it is necessary to choose one parameter to boost in each iteration. To do this we have to choose between parameters, and so we need to be able to find out which parameter would lead to the best increase in performance. We already do this for choosing which component-wise learner to use in each parameter. There, we choose that which has the best residual-sum-of-squares (RSS), with respect to the negative gradient vector. Thomas et al. (2018) denote this the *inner loss.* However, in general these generalized residual vectors are not comparable across parameters of the loss function, because the parameters have different scales (Thomas et al., 2018). In a normal distribution, for example, the partial derivatives for the mean and the partial derivative for the standard deviation will not be comparable. Therefore, to compare between parameters, a different comparison method is needed. We cannot compare the residual sums-of-squares, because they will not tell us which parameter will decrease the loss function the most. For each parameter θ_k, however, we choose the component-wise base learner which best fits according to the RSS,

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

as usual. If we incorporate this estimated base learner into the full boosted model, we would get

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

We can insert this proposed new model into the loss function to obtain a new empirical risk value,

$$R\left(\hat{\boldsymbol{\eta}}_k^{[m]}\right).$$

We calculate the gain in the loss function, which we denote $\Delta\rho_k$, by

$$\Delta\rho_k = R\left(\hat{\boldsymbol{\eta}}^{[m-1]}\right) - R\left(\hat{\boldsymbol{\eta}}_k^{[m]}\right),$$

where $\hat{\boldsymbol{\eta}}^{[m-1]}$, of course, is the current vector of additive predictors, from the previous iteration. For each parameter $k = 1, 2, \ldots, K$, we can calculate its ρ_k. If we now compare the gain in loss function value, we find out which parameter leads to the best increase. We choose that one, i.e.,

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

We incorporate only the best-fitting learner corresponding for that parameter into the full boosting model, i.e.,

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

meaning all components $k \in \{k\colon k \neq k^{[m]}, k = 1, 2, \ldots, K\}$ are not updated, so

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

3.11.2 Criterion for selecting component-wise learner

We choose the base learner for each component by comparing their residual sum of squares with respect to the negative gradient vector, which we called

the inner loss. In other words, we use the usual procedure of choosing the
component-wise learner for each parameter that minimizes the RSS, i.e.,

$$j^{[m]} = \underset{j}{\operatorname{argmin}} \sum_{i=1}^{N} (u_{k,i} - (f^{[m]}(x_i) + \hat{h}(\boldsymbol{x}_i)))^2.$$

This is, however, not the same criterion that is used to choose between param-
eters. This might be problematic. Therefore, Thomas et al. (2018) propose
using the loss function ρ for choosing between component-wise learners as well.
They call this the "outer loss." In the case of using the "outer loss," in each
parameter-specific step, we choose instead the component-wise learner that
minimizes the outer loss function, i.e.,

$$j^{[m]} = \underset{j}{\operatorname{argmin}} \sum_{i=1}^{N} \rho(y_i, f(x_i)).$$

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 models. E.g., by linear least squares if using simple
linear regression base learners. The improvement in the empirical risk, $\Delta\rho_k$,
is then calculated for each base learner of every distribution parameter, and
only the overall best-performing base learner with regard to the outer loss is
updated.

   In both cases of this algorithm, we have the advantage that the optimal
number of boosting steps, m_{stop}, is always a scalar value. 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.

   A schematic overview of the algorithm is given in algorithm 6.

Algorithm 6 Multidimensional noncyclical component-wise gradient boosting

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

$$\hat{f} = \underset{f}{\operatorname{argmin}} \, R(f).$$

2. Set $m = 0$. Initialize $f_1^{(0)}, f_2^{(0)}, \ldots, f_K^{(0)}$,, e.g., by setting it to zero for all components, or by finding the best constant.

3. Specify a base learner h_k for each dimension $k = 1, \ldots, K$.

4. Increase m by 1.

5. Set $k = 0$.

6. Increase k by 1.

7. 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

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

8. Fit the negative gradient vector to each of the p components of X (i.e. to each base learner) separately, using the base learners specified in step X. This yields p vectors of predicted values, where each vector is an estimate of the negative gradient vector $\boldsymbol{u}_k^{[m-1]}$.

9. Select the best fitting base learner, h_{kj}, either by

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

   $$j^* = \underset{j \in 1, \ldots, J_k}{\operatorname{argmin}} \sum_{i=1}^N (u_k^{(i)} - \hat{h}_{kj}(x^{(i)}))^2$$

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

   $$j^* = \underset{j \in 1, \ldots, J_k}{\operatorname{argmin}} \sum_{i=1}^N \rho\left(y^{(i)}, \hat{f}^{(m-1)}(x^{(i)}) + \nu \cdot \hat{h}_{kj}(x^{(i)}) \right)$$

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

$$\Delta \rho_k = \sum_{i=1}^N \rho\left(y^{(i)}, \hat{\boldsymbol{\eta}}^{[m-1]}(x^{(i)}) + \nu \cdot \hat{h}_{kj^*}(x^{(i)}) \right)$$

11. Update, depending on the value of the loss reduction, $k^* = \operatorname{argmin}_{k \in 1, \ldots, K} \Delta \rho_k$

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

while for all $k \neq k^*$,
$$\hat{\eta}_{k^*}^{[m]} = \hat{\eta}_{k^*}^{[m-1]}.$$

12. Repeat steps 4 to 11 until $m = m_{\text{stop}}$.

13. Return $\hat{\boldsymbol{\eta}}(\cdot) = \hat{\boldsymbol{\eta}}_M(\cdot) = \sum_{m=0}^M \boldsymbol{\eta}_m(\cdot)$.

Chapter 4

Multivariate component-wise boosting on survival data

In this chapter, we propose a component-wise boosting algorithm for fitting the inverse gaussian first hitting time model to survival data.

4.1 FHTBoost

The first-hitting-time model with Wiener processes, as shown, leads to inverse Gaussian lifetimes. As in the usual regression scheme (see subsection 2.5) for this setup, we we have $K = 2$ distribution parameters,

$$\boldsymbol{\theta}_i = (\theta_1, \theta_2)^T = (y_0, \mu)^T. \tag{4.1}$$

We choose the link functions

$$g_1(x) = \log(x) \tag{4.2}$$

and

$$g_2(x) = \text{identity}(x) = x, \tag{4.3}$$

for parameters y_0 and μ, respectively.

We wish to model the effect of the covariates on these parameters. In particular, and let y_0 by a high-dimensional matrix X, in practice typically gene expression data, and we will let μ be modeled by a low-dimensional matrix Z, typically clinical data. We let a vector of covariate information from one individual $i, i = 1, 2, \ldots, N$ be labeled $\boldsymbol{x}_i$ and $\boldsymbol{z}_i$, where these are gene data and clinical data, respectively. These matrices are defined as

$$X = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{p_1}), \tag{4.4}$$

and

$$Z = (\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_{p_2}), \tag{4.5}$$

where typically the number of dimensions p_1 of X is high (in particular, p_1 is much larger than n, $p_1 >> n$), and the number of dimensions p_2 of Z is relatively small, and in particular, $p_2 < n$.

We apply the GAMLSSBoost algorithm shown previously in Section 3.10 to this setup. The loss function of interest is the negative log likelihood of the censored inverse Gaussian distribution, derived in the Appendix, i.e.,

$$\rho(y, \boldsymbol{\theta}) = -l((t, d), (y_0, \mu)).$$

(See Appendix for the computational details!) We wish to build up an additive predictor $\boldsymbol{\eta}$, consisting of components η_1 and η_2. As in the GAMLSS setting, these are given by

$$\eta_1 := \exp(y_0) = \beta_{1,0} + \sum_{j=1}^{p_1} f_{1,j}(x_{1,j}), \tag{4.6}$$

and

$$\eta_2 := \mu = \beta_{2,0} + \sum_{j=1}^{p_2} f_{2,j}(x_{2,j}). \tag{4.7}$$

We wish to estimate these additive predictors by a gradient boosting algorithm. Given an estimated additive predictor $\hat{\boldsymbol{\eta}}$, we calculate the corresponding estimated distribution parameters by transforming the additive predictors via the inverse of their link functions. Thus,

$$y_0 = \theta_1 = g_1^{-1}(\eta_1) = \exp(\eta_1), \tag{4.8}$$

and

$$\mu = \theta_2 = g_2^{-2}(\eta_2) = \eta_2. \tag{4.9}$$

This means that as a function of the additive predictors, the loss function is

$$\rho(y, \boldsymbol{\eta}) = -l(\exp(\eta_1), \eta_2).$$

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 equal to the (positive) derivative of the (positive) log-likelihood function, with regard to each parameter. These are

$$-\frac{\partial}{\partial \eta_1}\rho(y_i, \boldsymbol{\theta}) = \frac{\partial}{\partial \eta_1}l(\exp(\eta_1), \eta_2) = f_1 \tag{4.10}$$

and

$$-\frac{\partial}{\partial \eta_2}\rho(y_i, \boldsymbol{\theta}) = \frac{\partial}{\partial \eta_1}l(\exp(\eta_1), \eta_2) = f_2. \tag{4.11}$$

See Appendix for the full details.

We can now apply the component-wise multidimensional boosting algorithm shown in the previous chapter. We choose to use the noncyclical variant as this seems to lead to equally good results and is less computationally intensive to tune, due to a one-dimensinal stopping parameter search, as stated in (Schmid et al., 2010).

More details here!

4.1.1 Initialization via maximum likelihood estimate

Recall the additive predictors,

$$\eta_1 = \beta_{1,0} + \sum_{j=1}^{p} f_{1,j}(x_{1,j}), \tag{4.12}$$

and

$$\eta_2 = \beta_{2,0} + \sum_{j=1}^{p} f_{2,j}(z_{2,j}). \tag{4.13}$$

We wish to estimate these by the boosting algorithm, but the boosting itself only provides the predictors $f_{k,j}$. To ensure proper estimation of η_1 and η_2, we need to compute the intercepts $\beta_{1,0}$ and $\beta_{2,0}$, 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 $\beta_{1,0}$ and $\beta_{2,0}$ only, namely

$$R(\beta_{1,0}, \beta_{2,0}) = \sum_{i=1}^{n} \rho(y, y_0, \mu), \tag{4.14}$$

where we use the discussed link functions,

$$y_0 = \exp(\beta_{1,0}), \tag{4.15}$$

$$\mu = \beta_{2,0}. \tag{4.16}$$

Hence the offsets are estimated to be

$$(\hat{\beta}_{1,0}, \hat{\beta}_{2,0}) = \operatorname*{argmin}_{\beta_{1,0}, \beta_{2,0}} R(\beta_{1,0}, \beta_{2,0}). \tag{4.17}$$

To do this in practice, we use the routine called `nlm`, which belongs to the base package of `R`.

Find citation for R programming language!

4.1.2 FHTBoost algorithm with fixed intercept

`algo:fhtboost`

1. Given a data set $D = \{\boldsymbol{x}_i, \boldsymbol{z}_i, t_i, d_i\}_{i=1}^{N}$, where for each observation i, $\boldsymbol{x}_i$ is a vector of clinical measurements, $\boldsymbol{z}_i$ a vector of gene expressions, t_i the possibly right-censored survival time, and d_i the censoring indicator. If $\boldsymbol{X}$ and/or $\boldsymbol{Z}$ are not normalized, do this. Standardize $\boldsymbol{X}$ and $\boldsymbol{Z}$ the loss function $\rho(y, \boldsymbol{\eta}(x, z))$, by proxy on the empirical risk, i.e., the loss function evaluated on the samples,

$$\hat{f} = \operatorname*{argmin}_{f} R(f). \tag{4.18}$$

2. Set iteration counter m to 0. Initialize additive predictors η_1 and η_2 to $\beta_{1,0}$ and $\beta_{2,0}$, respectively, by performing a numerical maximization of the likelihood, i.e.,

$$(\hat{\beta}_{1,0}, \hat{\beta}_{2,0}) = \operatorname*{argmin}_{\beta_{1,0},\beta_{2,0}} \sum_{i=1}^{n} \rho(y_i, \boldsymbol{\eta}). \qquad (4.19)$$

algostep:FHT-base-learner

3. Specify a base learner h_k for each dimension $k = 1, \ldots, K$. (We use linear least squares base learners.)

algostep:FHT-init

4. Increase m by 1.

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

$$\boldsymbol{u}_1^{[m-1]} = \left(-\frac{\partial}{\partial\hat{\eta}_1}\rho(y_i, \hat{\boldsymbol{\eta}}^{[m-1]}(x_i))\right)_{i=1}^{N} \qquad (4.20)$$

6. Fit the negative gradient vector to each of the $J_1 = p$ components of X (i.e. to each base learner) separately, using the base learners specified in step 3.

7. Select the best fitting base learner, $h_{1,j}$, by the inner loss, i.e., the RSS of the base-learner fit w.r.t the negative gradient vector

$$j^* = \operatorname*{argmin}_{j \in 1,\ldots,p} \sum_{i=1}^{N} (u_1^{(i)} - \hat{h}_{1,j}(x^{(i)}))^2. \qquad (4.21)$$

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

$$\Delta\rho_k = \sum_{i=1}^{N} \rho\left(y^{(i)}, \hat{\boldsymbol{\eta}}^{[m-1]}(x^{(i)}) + \nu \cdot \hat{h}_{1,j^*}(x^{(i)})\right) \qquad (4.22)$$

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

$$\boldsymbol{u}_1^{[m-1]} = \left(-\frac{\partial}{\partial\hat{\eta}_1}\rho(y_i, \hat{\boldsymbol{\eta}}^{[m-1]}(x_i))\right)_{i=1}^{N} \qquad (4.23)$$

10. Fit the negative gradient vector to each of the $J_2 = D$ components of Z (i.e. to each base learner) separately, using the base learners specified in step 3.

11. Select the best fitting base learner, $h_{2,j}$, by the inner loss, i.e., the RSS of the base-learner fit w.r.t the negative gradient vector

$$j^* = \operatorname*{argmin}_{j \in 1,\ldots,d} \sum_{i=1}^{N} (u_2^{(i)} - \hat{h}_{2,j}(x^{(i)}))^2. \qquad (4.24)$$

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

$$\Delta\rho_k = \sum_{i=1}^{N} \rho\left(y^{(i)}, \hat{\boldsymbol{\eta}}^{[m-1]}(x^{(i)}) + \nu \cdot \hat{h}_{2,\,j^*}(x^{(i)})\right) \tag{4.25}$$

13. Update, depending on the value of the loss reduction, $k^* = \operatorname{argmin}_{k \in \{1,2\}} \Delta\rho_k$

$$\hat{\eta}_{k^*}^{[m]} = \hat{\eta}_{k^*}^{[m-1]} + \nu \cdot \hat{h}_{k^* j^*}(x), \tag{4.26}$$

while for $k \neq k^*$,

$$\hat{\eta}_{k^*}^{[m]} = \hat{\eta}_{k^*}^{[m-1]}. \tag{4.27}$$

14. Repeat steps 4 to 13 until $m = m_{\text{stop}}$.

15. Return $\hat{\boldsymbol{\eta}}(\cdot) = \hat{\boldsymbol{\eta}}_{m_{\text{stop}}}(\cdot) = \sum_{m=0}^{m_{\text{stop}}} \boldsymbol{\eta}_m(\cdot)$.

4.2 Modification: Changing the intercept in each iteration

While developing the algorithm, I discovered that the algorithm as given above did not fully recover the known parameters. In particular, the estimated offset is not approximately equal to the original offset. In mboost, they seem to change the offset while boosting. This must surely be a problem others have encountered while deriving boosting algorithms.

Maybe cite a paper or R here.

4.2.1 FHTBoost algorithm with changing intercept

This algorithm is exactly the same as the previous one, algorithm 4.1.2, except for in step 13, where we numerically maximize the likelihood so as to find the optimal intercept:

1. Update, depending on the value of the loss reduction, $k^* = \operatorname{argmin}_{k \in \{1,2\}} \Delta\rho_k$

$$\hat{\eta}_{k^*}^{[m]} = \hat{\eta}_{k^*}^{[m-1]} + \nu \cdot \hat{h}_{k^* j^*}(x), \tag{4.28}$$

while for $k \neq k^*$,

$$\hat{\eta}_{k^*}^{[m]} = \hat{\eta}_{k^*}^{[m-1]}. \tag{4.29}$$

Find the best numerical constant to add to the intercept of the selected additive learner,

$$c = \operatorname*{argmin}_{c} \sum_{i=1}^{N} \rho\left(y^{(i)}, \hat{\boldsymbol{\eta}}^{[m]}(x^{(i)}) + c\right). \tag{4.30}$$

Add this c to the selected parameter

$$\hat{\beta}_{k^*} \leftarrow \hat{\beta}_{k^*} + c. \tag{4.31}$$

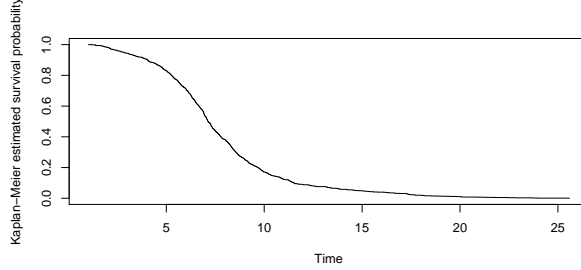Figure 4.1: Kaplan-Meier plot of generated survival data from subsection 4.2.2.

4.2.2   Example

We here simulate survival times from an inverse Gaussian FHT distribution. Let parameter vectors be $\boldsymbol{\beta} = (2, 0.1, 0.2)$ and $\boldsymbol{\gamma} = (-1, -0.1, 0.1)$. Let X and Z be such and such, drawn from a beta distribution. We simulate data using Algorithm 7 in section 6.2, with the censoring time W being drawn from a distribution $\exp(0.1)$.

> Fix this censoring distribution description.

The resulting survival times have the Kaplan-Meier plot shown in Figure 4.1. Figure 4.2 shows a plot of the negative log likelihood of the data (in-sample loss) as a function of the iteration number m. The solid black line shows the boosting method as given above, where we estimate the offsets $\beta_{1,0}$, $\beta_{2,0}$ before we begin iterating. These offsets are not changed in the boosting iterations. We use numerical maximization to obtain the joint maximum likelihood estimates of our data set. The offsets are $\hat{\beta}_{1,0}^{\mathrm{ML}} = 1.93$ and $\hat{\beta}_{2,0}^{\mathrm{ML}} = 0.18$. Running the boosting algorithm to convergence, in this case 35 iterations, gives estimates of the offsets as $\hat{\beta}_{1,0}^{\mathrm{ML}} = 1.69$ and $\hat{\beta}_{2,0}^{\mathrm{ML}} = 0.16$, respectively. We can see in the figure 4.2 that it quite clearly does not reach the maximum likelihood value, which is the solid blue horizontal line.

Modifying the algorithm to incorporate possible updates in the estimation of the intercept in each boosting iteration did make the algorithm reach maximum likelihood. To change the intercept in each boosting iteration, we can perform a new numerical maximization after each boosting step. This means to perform the same kind of numerical maximization as is done initially, at the beginning of each iteration, using now the estimated additive predictors as offsets. With this setup, we decompose the additive predictor into steps. We denote the initially estimated intercept $\hat{\beta}_{1,0}$ as $\hat{\beta}_{1,0}^{[0]}$. The intercept in the additive predictor now becomes a sum of boosted intercepts,

$$\hat{\beta}_{1,0} = \hat{\beta}_{1,0}^{[0]} + \sum_{m=1}^{m_{\mathrm{stop}}} \hat{\beta}_{1,0}^{[m]}, \tag{4.32}$$

where each $\hat{\beta}_{1,0}^{[m]}$ is a boost in the intercept estimated in step m of the boosting algorithm. This means that in iteration step m, we wish to estimate a boost $\hat{\beta}_{1,0}^{[m]}$. If we choose to boost parameter k, we change the intercept of η_k. If k is

Table 4.1: Parameter values of a model which reaches ML

	$\beta_{1,0}$	$f_{1,1}$	$f_{1,2}$	$\beta_{2,0}$	$f_{2,1}$	$f_{2,2}$
Maximum likelihood estimate	1.93	0.12	0.18	-0.94	-0.08	0.10
Fixed intercept	1.69	0.10	0.16	-0.71	-0.05	0.07
Changing intercept	1.92	0.11	0.17	-0.93	-0.07	0.09

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

1, we update the intercept for that parameter,

$$\hat{\beta}_{1,0}^{[m]} = \operatorname*{argmin}_c R(\eta_1^{[m]} + c, \eta_2^{[m]}), \tag{4.33}$$

and if k is 2, we update that intercept,

$$\hat{\beta}_{2,0}^{[m]} = \operatorname*{argmin}_c R(\eta_2^{[m]}, \eta_2^{[m]} + c). \tag{4.34}$$

The modified algorithm using iteratively updated intercepts is that in 4.2.1, with the step 13 substituted, as in subsection 4.2.1.

When using this algorithm with *changing* intercepts, the algorithm was successful in recovering the maximum likelihood value and parameters. See table 4.1 for the values, and graphically, in figure 4.2, where the convergence is plotted as a function of the number of boosting iterations.

Write a conclusion from the example: The algorithm with updating intercept seems better, as it should not be shrunk towards 0 (Hastie et al., 2009).

Figure 4.3: Negative log-likelihood for the boosting algorithm with both fixed
and iteratively changing intercept, as a function of iteration number m

fig:boosting-ML
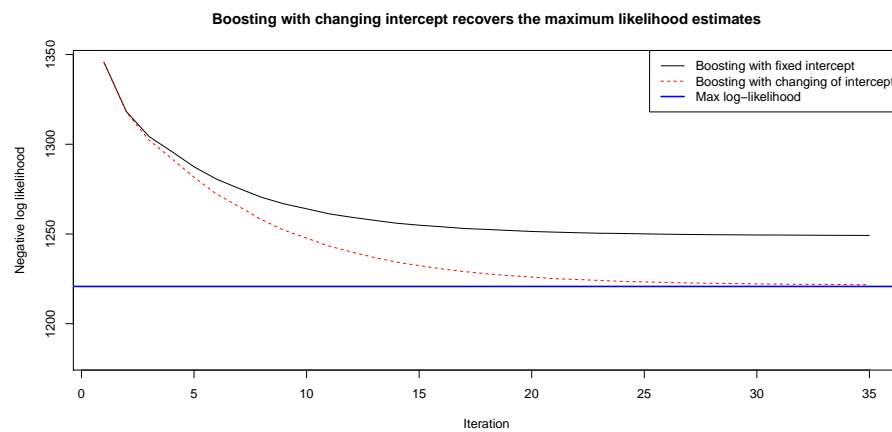
Chapter 5

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

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

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

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, $\hat{\boldsymbol{\theta}}_1$,

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

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:

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

and

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

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

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

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

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

and

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

Again let their concatenation be denoted

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

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

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

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

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.

$$\text{FDR} = \frac{FP}{FP + TP} \qquad (5.4)$$

`{eq:accuracy}`

5.3   Evaluating survival prediction with the Brier score

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

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

`{eq:phat}`

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

$$I(t_i > t^*). \qquad (5.6)$$

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

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

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 2.1.2). 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

$$BS^c(t^*) = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{\hat{\pi}(t^*|\boldsymbol{x}_i)^2 I(t_i \leq t^*, \delta_i = 0)}{\hat{G}(t_i)} + \frac{(1 - \hat{\pi}(t^*|\boldsymbol{x}_i))(I(t_i > t^*))}{\hat{G}(t^*)} \right].$$

(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 censored events having happened at or before time t^*, and the events that have not yet occurred.

5.3.3 Integrated Brier score

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_1 to time t_2. The integrated Brier score is simply the Brier score integrated over this time interval,

$$\text{IBS}(t_1, t_2) = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} BS^c(t) \, \mathrm{d}t.$$

In practice, we can obtain this by computing the Brier score 5.7 over a fine grid of time points $t \in [t_1, t_2]$, and then taking the average of these values.

By partitioning this time interval into smaller adjoint time intervals

$$t_{\text{start}} = t_0 < t_1 < \cdots < t_k = t_{\text{end}},$$

we can construct an approximation of the integrated Brier score by

$$\text{IBS}(t_{\text{start}}, t_{\text{end}}) \approx \sum_{j=1}^{k} BS^c(t_j) \cdot (t_{j+1} - t_j).$$

This needs work still

Chapter 6

Simulation study

After developing FHTBoost (see chapter 4), we wish to verify that it works, i.e. that it has predictive power and selects 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.

One 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, our first-hitting-time model), for which we know the true values of the parameters. We can then use our developed 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 model should fit the simulated data well, because the data generating mechanism *is* the model.

6.1   Simulation design

In this chapter we describe two scenarios, a highly correlated scenario and an uncorrelated scenario. The purpose of each scenario is to estimate FHT parameters, using the FHTBoost algorithm, on data simulated according to the scenario, and then to assess the model's performance on this data. In both scenarios, we generate a high-dimensional covariate matrix $\boldsymbol{X}$ consisting of covariate vectors

$$\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{p_1}, \tag{6.1}$$

which is supposed to mimic gene expression data. We also have a low-dimensional covariate matrix $\boldsymbol{Z}$, which similarly consist of covariate vectors

$$\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_{p_2}, \tag{6.2}$$

and which is supposed to mimic clinical measurements. We link each covariate to a parameter in a parameter vector, where $\boldsymbol{X}$ corresponds to $\boldsymbol{\beta}$, and $\boldsymbol{Z}$ corresponds to $\boldsymbol{\gamma}$. In each parameter vector, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$, we set only 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. Given these parameter vectors and covariate matrices, we can calculate a specific $y_{0,i}$ and a specific μ_i for each individual i, $i = 1, 2, \ldots, N$, 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 which represent the health process of each individual. To reiterate, an individual i has a stochastic health process, a Wiener process, with an initial level $y_{0,i}$ and a drift μ_i. Consequently its lifetime follows the lifetime probability distribution $\mathrm{IG}(y_{0,i}, \mu_i)$. This relationship allows us to easily draw a lifetime for each individual from its respective inverse Gaussian distribution.

It is important to evaluate the model/algorithm performance (both in terms of predictive ability and variable selection) on a separate and unseen test set. Each training set will be generated by drawing $N = 500$ individuals as described above, with a specific seed. Since we are simulating, it is simple to generate a test set by drawing from a unique seed. We are therefore also able to decide the size of the test set. We set $N_{\text{test}} = 1000$ observations.

We generate $B \approx 500$ data sets by drawing survival data according to algorithm 8, such that each data set has $N = 500$ observations. We treat each data set as a separate training data set, and thus estimate B models. To estimate each model, we first a perform repeated 5-fold cross validation procedure (see subsection 3.8.1), with 5 repeats, on the training data set. As shown in subsection 3.4.4, this should provide a reasonably variance reduced estimate of m_{stop} (near the "true" m_{stop}). We then estimate a model on the training set, by running FHTBoost with m_{stop} number of iterations.

The boosting algorithm has two main purposes: Selection of variables, and minimizing test error. To assess variable selection, we look at some well known metrics. To assess the test/generalization error, we finally calculate the difference of deviance on the test set.

We perform simulations in which the observations are drawn from the inverse Gaussian distribution, i.e., we simulate lifetimes from the first hitting time model with Wiener process as the health process. We do this by using algorithm 7. To simulate the covariate matrices X and Z we will use Algorithm 8, which is a method for simulating clinical and gene data together. We imagine X, corresponding with regression parameter vector $\boldsymbol{\beta}$, be gene expressions, whereas Z, which is corresponding to the vector of regression coefficients $\boldsymbol{\gamma}$, be clinical measurements. We specify the different correlations for the covariate matrices. But most importantly, we specify the true parameter vectors, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$. For each scenario, we conduct N_{scenario} runs.

One run consists of first drawing data (with a specific seed to ensure reproducibility), i.e., we draw covariate matrices $\boldsymbol{X}$, $\boldsymbol{Z}$ from Algorithm 8. $\boldsymbol{X}$ is of size $n \times p$ and $\boldsymbol{Z}$ is of size $N \times d$, where N is the number of observations, $p + 1$ is the size of the covariate vector $\boldsymbol{\beta}$ (including an intercept which will not be affected by the covariates), and $d + 1$ is the size of the covariate vector $\boldsymbol{\gamma}$. Then we combine these with the true covariate matrices to get vectors $\boldsymbol{y}_0$ and μ of initial value of the health process, and drift, respectively. Then we draw from the Inverse Gaussian distribution according to Algorithm 7, obtaining N right-censored lifetimes, i.e., N tuples $(t_i, d_i)_{i=1,\ldots,N}$. With these tuples, then, we can do a run with the FHT boosting algorithm. We first use repeated K-fold cross-validation to find the optimal number of boosting steps, m_{stop}. Then we estimate the model on the whole of this training set. Then we validate this model on a test set of size N_{test}. The data here are drawn in the exact same manner as the training data, again with a specific seed for reproducibility.

6.2 Simulation of survival data from an IG FHT distribution

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.

To censor the data, we draw censoring times $w_i \sim f(\cdot), i = 1, \ldots, N$ from some other lifetime distribution where the parameters do *not* depend on covariates. We let the observed survival times then be $t_i = \min(\tilde{t}_i, w_i)$. 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 set of N tuples $(t_i, d_i), i = 1, \ldots, N$. Note that this prcedure simulates censored time under independent censoring, since indeed the censoring times are independent of the survival times.

Algorithm 7 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 $\boldsymbol{\gamma}$.

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 $\mathrm{IG}(\mu, y_0)$.

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

To create a realistic scenario where we have data looking like gene expression data and clinical data, we need to define a proper correlation structure. We can draw covariate matrices from a normal distribution with a suitable covariance matrix.

We consider a scenario where we have a covariate matrix X consisting of p gene expressions, and a covariate matrix Z consisting of d clinical measurements. We can imagine that some of the genes in X are highly correlated. One way to imagine this is to imagine that we have blocks of genes, where inside one block, the genes are highly correlated, whereas genes in one block are not correlated to other genes. In addition, one block of genes might affect a block of clinical variables as well.

We specify a number of blocks B. A given block $b, b = 1, 2, \ldots, B$, contains a certain number of genes, G_b, which are correlated to each other. It also contains a certain number of clinical measurements, C_b. These measurements are correlated to each other, and to the genes in the block.

After setting up the block structure, we

Finish this

There are three types of correlations. 1. Within each block of genes. Defaults to 0 for genes not belonging to any block. 2. Between clinical predictors in each pathway 3. Between the clinical and molecular predictors in each pathway

Algorithm 8 contains a schematic overview.

---

algo:clinical-sim

Algorithm 8 Generating correlated clinical and gene expression data

1. Lorem ipsum.

6.4   Scenarios

6.4.1   Scenario 1: Uncorrelated case

We generate a data set of size $N_{\text{scenario}} = 500$, and parameter vectors of size $10000 + 1$ and $15 + 1$, respectively. We let $\boldsymbol{\beta}$ be a large vector of size $p = 10001$, and $\boldsymbol{\gamma}$ be a small vector of size $d = 16$. We first discuss the size of the parameter effects. 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 there was much signal from the underlying parameter vector. 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 be 2.0, and the first 35 elements to be 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

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

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

We draw X and Z from Algorithm 8 for drawing clinical and gene data, with $B = 0$ blocks.

We specify that all correlations are 0, meaning no covariate correlates with any other.

With this exact setup, we run a simulation experiment $B = 500$ times, where we set the seed at the beginning of each simulation. We first generate matrices X and Z from Algorithm 8, with all correlations set to 0. Based on these covariate matrices, we simulate FHT lifetimes from Algorithm 7. Having now a data set D_b, we run cross validation on this data set to find the optimal iteration number m_{stop}. See Figure 6.1 We then run a boosting algorithm with m_{stop} steps on the training set, to estimate parameters.

6.4.2 Scenario 2: Correlated

Consider now a scenario where we have correlation in different ways. We have blocks of genes which are correlated, meaning the genes comprising one are correlated. Further, the genes in a block of genes are correlated with certain clinical measurements. Finally, the clinical measurements are correlated. All these correlations are set to 0.7. In other words, this is a scenario with highly correlated data.

There are three types of correlations. 1. Within each block of genes. Defaults to 0 for genes not belonging to any block. 2. Between clinical predictors in each pathway 3. Between the clinical and molecular predictors in each pathway

6.5 Results

After having estimated an FHT model, we calculate the difference of deviance of the estimated model on the test set (see Section 5.1). In addition, 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 6.1: Difference of deviance results for FHTBoost, uncorrelated case

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

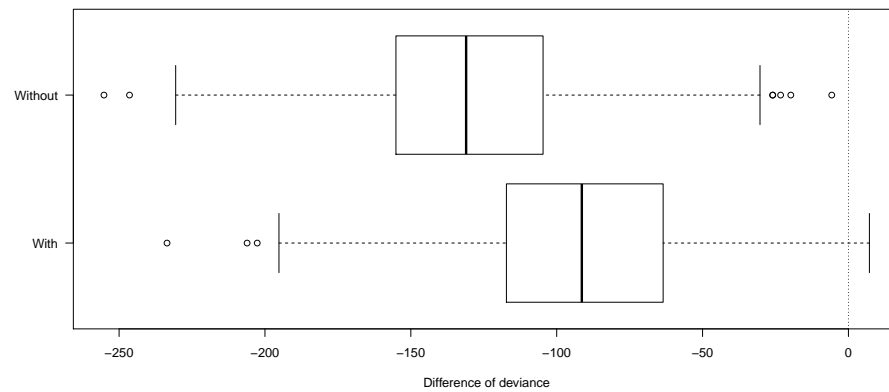Figure 6.1: Boxplot for difference in deviance for the two intercept variants, non-correlated scenario

6.5.1   Uncorrelated case

6.5.1.1   Test set performance/model fit

The most evident result is that, in contrast to our expectation, the fixed intercept algorithm seems to perform on average better than that with updating intercept (see Table 6.5): 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 sd in Table 6.5), this difference is 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 6.5, row "min", and Figure 6.1). In other words, for the fixed intercept version, all estimated models 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 now consider the variable selection metrics.

6.5.1.2   Variable selection

Let us contrast the two versions of FHTBoost in terms of variable selection as well. This scenario has 35 informative gene covariates and 5 informative clinical covariates. 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

Table 6.2: High dimensional (genomic) part: Performance of FHTBoost in terms of variable selection, uncorrelated case

	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: uncorrelated-y0

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

	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: uncorrelated-mu

Table 6.4: Optimal iteration number m_{stop} results for FHTBoost, uncorrelated case

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

table: uncorrelated- mstop

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.

Write this better!

Note also that the two versions of the algorithm have very different m_{stop} values. The stopping iteration m_{stop} when continuously changing the intercept is quite lower than when boosting without. This means that the resulting covariates for the changing intercept version are more shrunken than for the fixed intercept version. This suggests that the non-fixed intercept version is quicker to overfit, since the m_{stop} should find approximately the iteration number which minimizes overfitting. 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.

Furthermore, we are considering the sensitivity and specificity on both covariate vectors at the same time. This means that these scores will affect each other.

Consider first the result of the version in which the intercept is changed in each step. Both covariate vectors have a very high specificity, which measures the amount of true negatives which are correctly classified as negatives, i.e., not selected. The changing intercept version has a mean specificity of 1.00.

Table 6.5: Difference of deviance results for FHTBoost, uncorrelated case

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

Figure 6.2: Boxplot for difference in deviance for the two intercept variants, non-correlated scenario
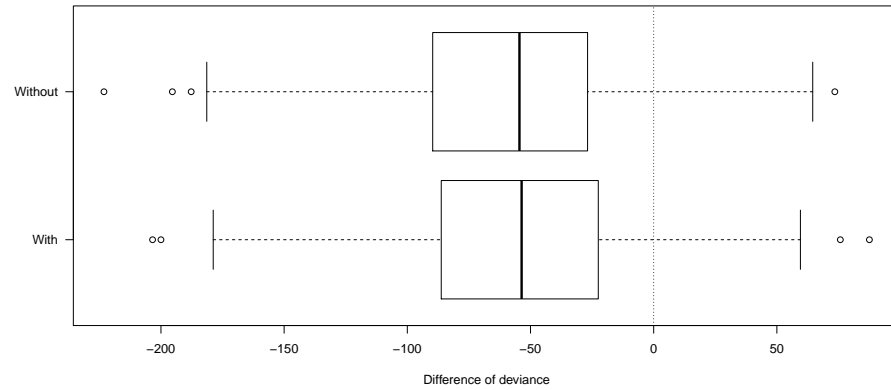
With a mean false discovery rate of 0.316, there is about a one in three chance that we select a variable which is not actually informative. The sensitivity, i.e., the ratio of correctly selected informative variables, has a mean of only 0.190. This means that a large proportion of the informative covariates are not selected. For γ, 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 β covariate vector, a higher proportion of informative covariates are selected, with a mean sensitivity of 0.452. The mean specificity is 0.997. Simultaneously, a larger mean false discovery rate of 0.613 is not good: This means that more than half should be expected to be false. At the same time, the γ covariate vector has a really good sensitivity, with a mean of 0.958, and a good specificity with a mean of 0.638. The false discovery rate on γ is slightly above one in three, with a mean of 0.375.

6.5.2   Correlated case

6.5.2.1   Test set performance/model fit

We see that the median deviance is very slightly better for the fixed intercept version, and with better extreme values, both minimum and maximum. See Figure 6.2 for a boxplot of the difference of deviance for the two versions.

Table 6.6: High dimensional (genomic) part: Performance of FHTBoost in terms of variable selection, correlated case

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

	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)

6.5.2.2 Variable selection

Now consider the variable selection metrics. We first look at the covariate vector β, which affects the initial level y_0. The fixed intercept version is slightly better with regards to sensitivity, i.e., selecting informative variables, with a mean of 0.204 versus a mean of 0.157 for the changing intercept version. This does come at a cost of a higher false discovery rate, with a mean as high as 0.652, where the changing intercept version has a mean of 0.439. This means that we should expect that more than two in three In both versions, then, it is at least as likely to select a non-informative variable as an informative variable. The specificity score is almost perfect in both cases, but again we note that the denominator N is very large.

Look now at the "clinical" covariates, used in γ and related to the drift μ. The changing intercept version performs quite a lot worse here than the fixed intercept version. It achieves a mean sensitivity of 0.273, while the fixed intercept version achieves 0.625. This does come at a cost of a lower specificity, with a mean of 0.537 for the fixed intercept and 0.831 for the changing intercept. The mean of the fixed intercept is slightly higher, at 0.507, whereas the changing intercept version has a mean of 0.454.

For this scenario, as well, we see 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.

In conclusion, while the fixed intercept version also here selects more true positive variables, it comes at a cost of selecting more false positives. If we use the deviance to assess which one is best, then the fixed intercept version is the better one here as well.

Table 6.8: Optimal iteration number m_{stop} results for FHTBoost, correlated case

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

table: correlated-mstop

6.6   Discussion

Based on the results in this simulation study, we conclude that it is better to use the fixed intercept version of the algorithm. It has performed consistently, but only slightly better with regards to log-likelihood/deviance on out-of-sample data. 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 generalization error in the form of a lower difference of deviance.

Make clear that in the correlated case we are more close to real data, and here the differences are not huge!

Chapter 7

Application on real data

In the simulation we contrasted fixed/updating intercept versions, and in general, both FHTBoost seems to provide a decent model for correlated, realistic survival data, with average deviance much smaller than 0.

Make this paragraph better (merge)

To truly be applicable in biomedical settings, we must see if the algorithm manages to achieve predictive power in a real study. In this chapter, we consider data from Oberthuer et al. (2008), consisting of data from patients diagnosed with neuroblastoma. The data consists of a small number of patients, with information on two covariates and around 10 000 gene expressions. We use FHTBoost on this data and assess its performance. We also compare the predictive power of our method to Cox regression, specifically CoxBoost, a version of Cox which has been adapted to a boosting framework (see, e.g., Binder and Schumacher (2008)), i.e., much like FHTBoost.

7.1 Neuroblastoma

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 range from a wait-and-see approach for those in the low risk group, to intensive treatment for the high-risk group. Still, common clinical experience suggests that such risk classification is still suboptimal for a substantial number of patients. The individual courses within these risk groups, in particular those of high-risk patients, still vary clearly.

Originally, the data consists of two separate data sets: A larger training set, collected from Germany, and a smaller test set, collected from 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 centers in several countries (including Germany). In this set, 29 of the samples were obtained from German patients enrolled in German neuroblastoma trials, while the remaining samples are from patients enrolled in national trials in other countries.

Due to few 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 which 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 possibly censored survival time. This survival time was defined as time from diagnosis to first recurrence, and was censored at 5 years. In addition to the aforementioned risk group, the patient's age at diagnosis is recorded, resulting in two clinical covariates per patient. 89 out of the 362 observations have a missing age. We remove all of these observations, and are left with a data set of 273. So $N = 273$. Of these 273, 86 children were classified as having high-risk, while the remaining 187 were not. It is coded as a binary variable, where both low risk and intermediate risk are coded as 0, and high risk as 1. 42 of the 273 children experienced a recurrence within the follow up, a proportion of 31.5%.

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, and generate 100 random splits times. First, however, we report the analyses of one single (the first) split of train and test data, to give an example of how to interpret the results.

7.2   Single split

We standardize the data to ensure FHTBoost works. The data are then 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.

7.2.1   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 estimate of m_{stop}. We perform a (10 times) repeated 5-fold cross validation to find the optimal number of iterations. Note in Figure 7.1 the impact of running the 5-fold cross-validation in a repeated fashion: Had we used a single 5-fold cross-validation, we would have selected different m_{stop}. Had either of these been a s the optimal m_{stop} is different between these different runs. We first performed a 10-fold repeated cross-validation, but this parameter search did not converge, i.e., the log-likelihood kept increasing. Upon further inspection, we found that one of the folds was the main cause of this, as the log-likelihood for that particular fold continued to increase, even after 400 iterations, while the other folds were in overfitting territory. We concluded that splitting a training set of around 180 into 10 folds would eventually cause a problem in one of the folds, as there was too little information left. We find $m_{\mathrm{stop}} = 20$ to be optimal in this case,

Figure 7.1: Repeated 5-fold cross validation on training set generated from neuroblastoma data set (Oberthuer et al., 2008).
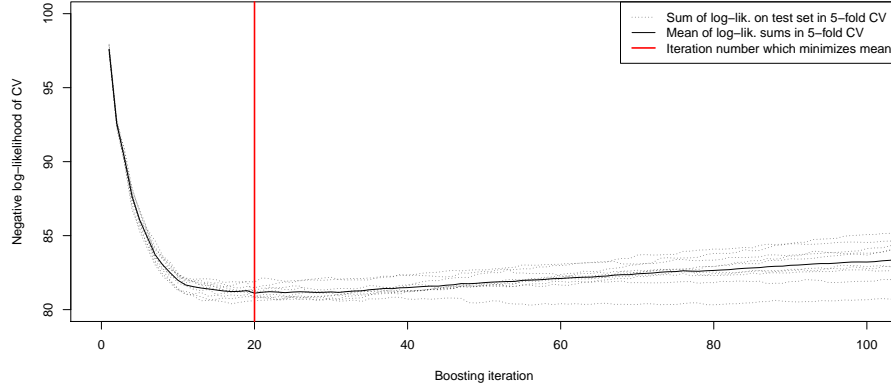
Table 7.1: Estimated null model on neuroblastoma (Oberthuer et al., 2008)

	Value
β_0	0.692
γ_0	0.077

as shown in Figure 7.1. Each dotted gray line 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 iteration number. The solid black line is the mean of these 10 gray lines, and the red vertical line indicates the optimal m_{stop}, i.e., the minimizing iteration number.

7.2.2 Results

We run the boosting algorithm with m_{stop} iterations, to estimate the model parameters on the entire training set.

7.2.2.1 Baseline, null model, interpretation

We first look at the intercepts, reported in Table 7.1, meaning, the null model parameters. The estimated intercept for the gene data, β_0, is 0.692. We recall that in the FHT model, the vector $\boldsymbol{\beta}$ corresponds to the initial level y_0 of the health process, with the log link function. The null model, without any covariate effects, therefore has a y_0 of $\exp(0.692) = 1.998$. Further, the intercept for the clinical data is estimated to be 0.077. 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 1.998, and with a *positive* drift, albeit slightly, of 0.077. Recall also that the Wiener process has a unit variance in each time unit, meaning the variance is equal to the time unit t. The large variability of the process relative to its starting point means there is still a significant chance

Figure 7.2: Wiener processes with parameters $y_0 = 1.998$ and $\mu = 0.077$, corresponding to the estimated null model from the neuroblastoma data set (Oberthuer et al., 2008).

fig:
neuroblastoma-
wien
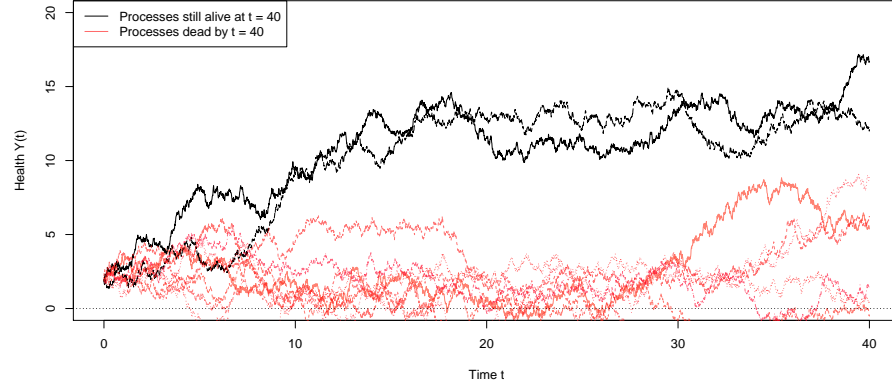
of recurrence of neuroblastoma. The resulting health process is

$$Y(t) = 1.998 + W(t) \cdot 0.077t,$$

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

$$Y(t) \sim N(1.998 + 0.077t, \sqrt{t}).$$

To get a feeling of the variability of it, and potential trajectories of such a process, we plot 10 realizations of this process in Figure 7.2. Of these particular 10 processes, 8 processes at some point go below 0. In the FHT interpretation, then, these health processes would cause a death, or, a recurrence of the neuroblastoma cancer. To get a better estimate of the proportion of health processes not dying, we need to sample more processes. We sampled 10000, and 6221 of these went below 0 within $t = 40$, i.e., a proportion of 0.378 did not experience recurrence. In section 2.3, specifically in equation (2.20), we stated 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 we have here in our null model. We calculate this for our estimated null model, obtaining

$$\Pr\left(T = \infty\right) = 1 - \Pr\left(T < \infty\right) = 1 - \exp\left(-2 \cdot y_0^{[0]} \cdot \mu^{[0]}\right)$$
$$= 1 - \exp\left(-2 \cdot 1.998 \cdot 0.077\right) = 0.265,$$

meaning about three in four should have a recurrence during their lifetime. For this training set, 28 out of 182 are observed events, meaning only 0.154 have experienced a recurrence. Note that this is with a medium follow-up of 4.5 years. It is, however, still a bit off from the cure rate obtained from the estimated null model, but our model does predict a non-zero proportion of long-term survivors, which is good.

Consider the fact that the initial level y_0 is quite low. Since we are looking at when, if ever, a neuroblastoma patient will have a recurrence after a surgery, this

initial level of the Wiener process is the health level of a patient at surgery. The health of such a child is often presumable quite precarious, as neuroblastoma is a malignant cancer. Therefore it makes sense that the initial level y_0 is quite low. However, the survival probability of patients vary, as is experienced by clinicians, even among those specified as high-risk. Most children will, as estimated by our model, survive.

Now consider the fact that the drift μ is positive. 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 the surgery.

7.2.2.2 Estimated covariates

Let us further look at estimated covariate effects (Table 7.2 and Table 7.3) We see that the boosting algorithm has included both covariates into the model. Before splitting into training and test sets, we centered and scaled each column of the covariate matrices, such that the mean across all individuals is (approximately) 0 for each covariate j, and the standard error is approximately 1. 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 corresponding to the clinical measurements, we should scale these back to 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 are -0.677 and 1.472, respectively. The most striking result here is the large parameter corresponding to risk. We calculate the drift parameter for those individuals designated as high-risk, and it is

$$\mu_{\text{high-risk}}^{[0]} = 0.077 - 0.189 \cdot 1.472 = -0.202.$$

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

$$\mu_{\text{low-risk}}^{[0]} = 0.077 - 0.189 \cdot -0.677 = 0.205.$$

Since age is standardized, these two drift covariates should be the mean drift parameters for each of the 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

$$-0.029 \cdot 4.193 = -0.122.$$

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

$$-0.029 \cdot 0.724 = 0.021.$$

Hence, crucially, this means that the model predicts that all high-risk individuals will eventually have a recurrence of neuroblastoma cancer. This seems to resonate with the fact that these children are indeed characterized as having

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

Gene j	β_j (full)	β_j (genomic only)
Gene 49	-0.010	0
Gene 1447	-0.069	-0.047
Gene 3191	0	-0.051
Gene 2442	0.012	0
Gene 4447	0	-0.062
Gene 5307	0	0
Gene 5527	-0.073	-0.098
Gene 5725	-0.009	0
Gene 6532	-0.011	0
Gene 6701	0.015	0
Gene 6901	0.011	0

Table 7.3: Results of estimated clinical coefficients on neuroblastoma data (Oberthuer et al., 2008).

Clinical covariate	γ_j (full)	γ_j (clinical only)
Risk	-0.189	-0.268
Age	-0.029	0

a high risk of recurrence. Those not designated as high-risk, on the other hand, have a probability of not experiencing recurrence, using the formula seen previously, namely

$$1 - \exp\left(-2 \cdot y_0^{[0]} \cdot \mu_{\text{low-risk}}^{[0]}\right) = 1 - \exp\left(-2 \cdot 1.998 \cdot 0.205\right) = 0.559.$$

We should therefore expect, based on the estimated model, that more than half of the low-risk patients should recover.

We observe that 8 genes have been selected in 20 boosting iterations (see Table 7.2). Some effects are estimated to be positive, some to be negative. 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 that still depends on the distribution of these. It is more difficult here to say how much the genomic data helps in explaining the variance.

7.2.3　Difference of deviance on the test set

The test set consists of 91 individuals, of which 14 have experienced recurrence. With the estimated model, we calculate the deviance, as seen in Section 5.1. Recall that 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 on -95.2, which is quite a lot!

Table 7.4: Difference of deviance results.   `tab:deviances`

Boosting type	Difference of deviance
Full	-95.2
Clinical (y_0) only	-14.5
Genetic (μ) only	-104.4

7.3 Comparing a clinical-genetic model to clinical-only and genetic-only models

Bøvelstad et al. (2009) analysed the neuroblastoma data and compared different ways of combining clinical and genomic data in Cox models. As mentioned in subsection 2.5.3, the FHT model lends itself easily to combining genomic and clinical data. Our boosting method FHTBoost offers a simple way to combine clinical and genetic data in estimating. There is also a straightforward way to only use clinical covariates, or to only use genetic covariates. To do this, we can use the cyclical version of the algorithm, where we booth both parameters in each step, but they have their own tuning parameter. This lets us fix the number of boosting steps of the parameter not to be boosted to 0. In other words, a genomic version of FHTBoost, or y_0-only version of FHTBoost, where we boost only 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).

We do the estimation and the test set calculation, and we get the difference of deviance results shown in table 7.4. We see here, in fact, that in this case the full model is beaten by the genomic model, with difference of deviance of -95.2 and -104.4, respectively.

Furthermore, we calculate the Brier scores at the time points of the test set. See Figure 7.3 for a comparison of the Brier score of the three different FHT models. We see here that, in fact, the genomic model performs best, and quite strikingly so.

7.4 Comparison with the Cox model

The Cox model which has been discussed previously, in Chapter 2. We compared our model with the Cox model as implemented in CoxBoost (**?**), setting

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

`{eq:lambda-nu}`

as suggested in De Bin (2016), where 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.

For the specific case discussed previously, we calculate the Brier score for these models. Consider now a comparison between the Brier score for the full FHT model and the Cox model, in Figure 7.4. We can see that the Cox model
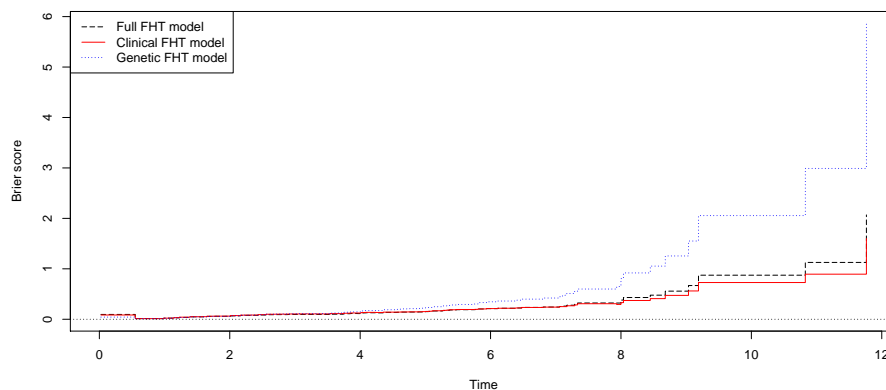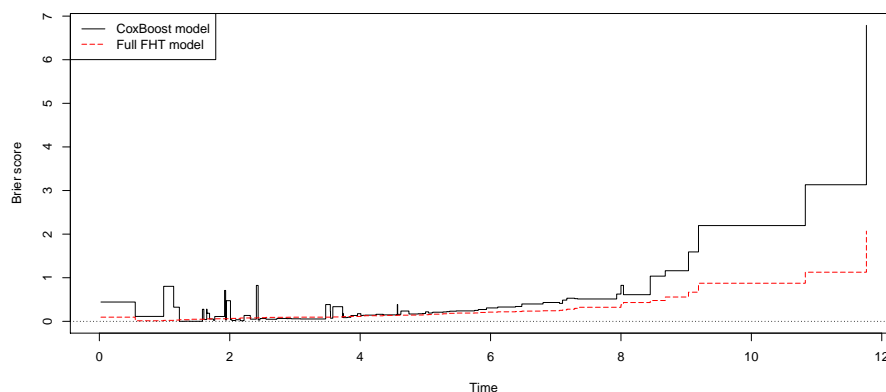
Figure 7.3: Brier scores for FHT models.

Figure 7.4: Brier scores for Cox and both.

performs better for all times, but the difference in performance also increases over time. Consider now Figure 7.5, in which we compare the Cox model to the genomic model. What is especially interesting here is that the Brier score of the genomic model almost overlaps with the Brier score of the Cox model. That is, except the positive jumps that the Cox model performs in the beginning, mostly from time 0 to 4. I do not know why these jumps happen.

See Figure 7.4 for a comparison of the Brier score of the boosted Cox model and the boosted FHT model.

7.5 Analysis of 100 train/test splits

Bøvelstad et al. (2009) generated 50 random splits of training and test sets from the data, to see the distribution. We now generate 100 splits of training and test sets, in the same manner. We use the same method as above. We first estimate parameters based on the training set, and then calculate the model's

Figure 7.5: Brier scores for Cox and genetic FHT model.

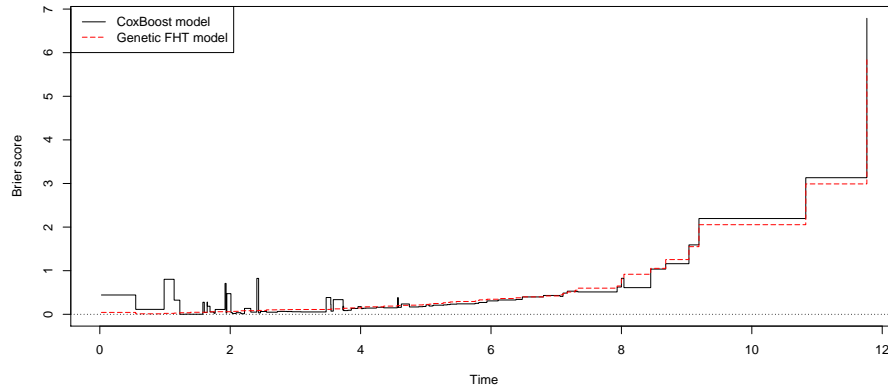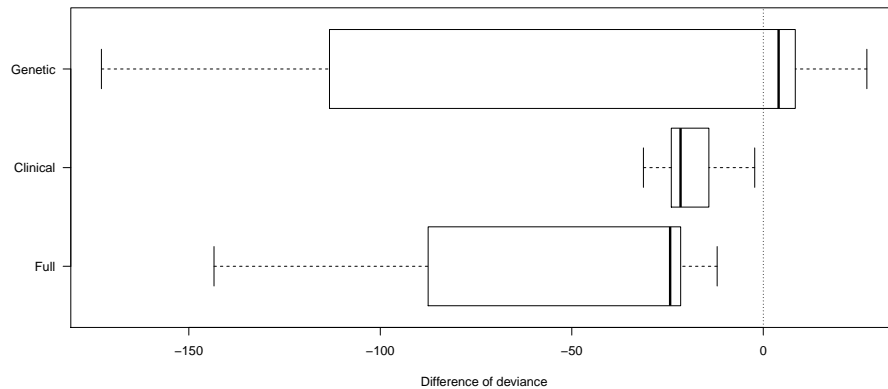Figure 7.6: Boxplot for difference in deviance for different variants of the FHT model.

difference of deviance, on the test set, using the parameters estimated on the training set.

7.5.1 Comparing deviance of FHT models

We now consider the difference of deviance across all 100 splits of training and test sets. All The See Figure 7.6 for difference of deviance boxplot. We use the median of difference of deviance as the main measure of interest. It is -24.3172 for the full model, and -21.60033 for the clinical model, and 4.009162 for the genomic model. See Figure 7.6 for a boxplot of these.

These deviances look a bit strange, at least the box for the full model, and the box for the genomic model. The median looks to be very strangely positioned, all the way to the right of the boxes. We plot histograms of these two to consider why. See Figure 7.7. The reason turns out to be that they

Figure 7.7: Histogram of difference of deviance for the genomic model and the full model.
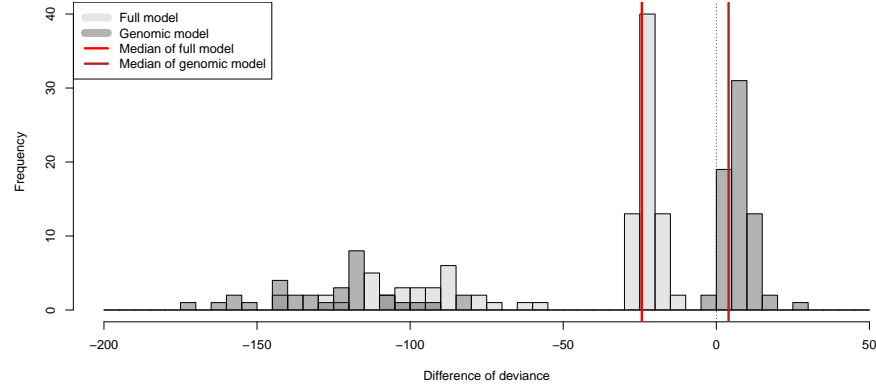
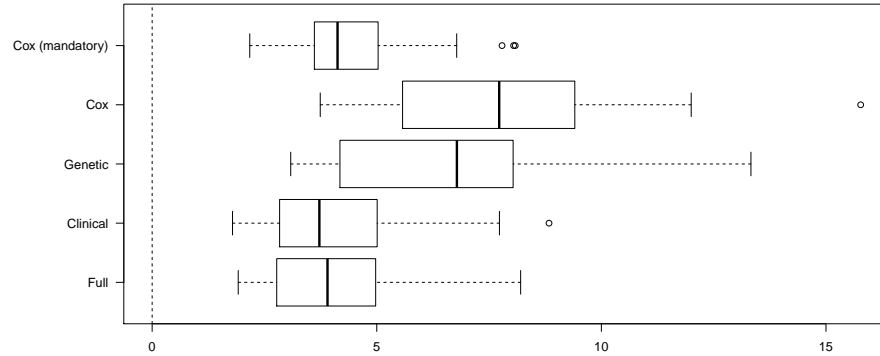Figure 7.8: Boxplot of integrated Brier scores.

are quite bimodal in their distribution. Both have large peaks around 0, and both have smaller and wider peaks more to the left. We see that the full model consistently improves performance with covariates, i.e., has a negative difference of deviance. This resonates with the fact that the clinical model also improves over the null model. However, the genomic model very often does not improve on the null model. Although, its most extreme values are in cases where the difference of deviance is very small, and so these are cases where the genomic model outperforms the full model.

7.5.2   Integrated Brier scores results

We now calculate integrated Brier scores for all 100 splits. A boxplot can be seen in Figure 7.8. With regard to this integrated Brier score, the full model performs slightly better than the clinical model. It turns out that the full model

performs slightly better than the clinical model, based on the median difference of deviance. It might therefore appear as if the genomic data sometimes improves performance, and sometimes does not. However, the genomic-only model vastly outperforms the full FHT model. We get a median integrated Brier score of 3.9 for the full FHT model, only slightly beating the clinical one at 3.7. The genomic model, however, achieves 6.8, whereas the Cox model performs best, at 7.7.

I'm not sure why this happens, especially since the log-likelihood of the full model is best. It looks like the observations which improve the likelihood are not those which improve the Brier score. If these were the same, then we would most likely see the same model perform best in both regards. It might, for example, be, at least based on the example seen earlier, that the genomic model in this case better explains the later observations, whereas the clinical data is better at lifetimes of smaller value.

7.6 Conclusion

Applying the FHT model to this data set has been a partly fruitful effort. The model is able to incorporate covariate information to improve the model fit. However, the predictive power is quite off from the Cox model.

Appendices

Appendix A

Appendix 1: Differentiating the IG FHT

First we have the likelihood,

$$L(y_0, \mu) = \prod_{i=1}^{n} \left(\frac{y_0}{\sqrt{2\pi\sigma^2 t_i^3}} \exp\left[-\frac{(y_0 + \mu t_i)^2}{2\sigma^2 t_i} \right] \right)^{\delta_i}$$
$$\times \left[1 - \Phi\left(-\frac{y_0 + \mu t_i}{\sqrt{\sigma^2 t_i}} \right) - \exp\left(-\frac{2y_0\mu}{\sigma^2} \right) \Phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}} \right) \right]^{1-\delta_i}, \qquad \text{(A.1)} \quad \boxed{\texttt{\{eq:fht-lik\}}}$$

with respect to parameters μ, and y_0. First, note that for any cumulative distribution function F that is symmetric around 0, and for $x \in \mathbb{R}$,

$$F(x) = 1 - (1 - F(x)) = 1 - F(-x), \qquad \text{(A.2)}$$

and so in particular,

$$\Phi(x) = 1 - (1 - \Phi(x)) = 1 - \Phi(-x), \qquad \text{(A.3)}$$

and thus we can rewrite (A.1) as

$$L(y_0, \mu) = \prod_{i=1}^{n} \left(\frac{y_0}{\sqrt{2\pi\sigma^2 t_i^3}} \exp\left[-\frac{(y_0 + \mu t_i)^2}{2\sigma^2 t_i} \right] \right)^{\delta_i}$$
$$\times \left[\Phi\left(\frac{y_0 + \mu t_i}{\sqrt{\sigma^2 t_i}} \right) - \exp\left(-\frac{2y_0\mu}{\sigma^2} \right) \Phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}} \right) \right]^{1-\delta_i}. \qquad \text{(A.4)} \quad \boxed{\begin{array}{c}\texttt{\{eq:fht-lik-}\\\texttt{proper\}}\end{array}}$$

It is easier to work with the log likelihood, so we take the log of (A.4) and get

$$l(y_0, \mu) = \sum_{i=1}^{n} \delta_i \left(\ln y_0 - \frac{1}{2} \ln\left(2\pi\sigma^2 t_i^3\right) - \frac{(y_0 + \mu t_i)^2}{2\sigma^2 t_i} \right)$$
$$+ (1 - \delta_i) \ln\left(\Phi\left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}} \right) - \exp\left(-\frac{2y_0\mu}{\sigma^2} \right) \Phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}} \right) \right) \qquad \text{(A.5)} \quad \boxed{\texttt{\{eq:fht-loglik\}}}$$

To make things easier, let us introduce some intermediate functions here. Let

$$\ln f_i(y_0, \mu) = \ln y_0 - \frac{1}{2} \ln\left(2\pi\sigma^2 t_i^3\right) - \frac{(y_0 + \mu t_i)^2}{2\sigma^2 t_i} \qquad \text{(A.6)}$$

and

$$S_i(y_0, \mu) = \Phi\left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}}\right) - \exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right). \qquad \text{(A.7)}$$

So we get

$$l(y_0, \mu) = \sum_{i=1}^{n} \delta_i \ln f_i(y_0, \mu) + (1 - \delta_i) \ln S_i(y_0, \mu) \qquad \text{(A.8)}$$

Thus we see that the partial derivatives are

$$\frac{\partial}{\partial y_0} l(y_0, \mu) = \sum_{i=1}^{n} \delta_i \frac{\partial}{\partial y_0} \ln f_i(y_0, \mu) + (1 - \delta_i) \frac{\frac{\partial}{\partial y_0} S_i(y_0, \mu)}{S_i(\theta)} \qquad \text{(A.9)}$$

and

$$\frac{\partial}{\partial \mu} l(y_0, \mu) = \sum_{i=1}^{n} \delta_i \frac{\partial}{\partial \mu} \ln f_i(y_0, \mu) + (1 - \delta_i) \frac{\frac{\partial}{\partial \mu} S_i(y_0, \mu)}{S_i(\theta)} \qquad \text{(A.10)}$$

We take these one by one

$$\frac{\partial}{\partial y_0} \ln f_i(y_0, \mu) = \frac{1}{y_0} - \frac{y_0 + \mu t_i}{\sigma^2 t_i} \qquad \text{(A.11)}$$

$$\frac{\partial}{\partial \mu} \ln f_i(y_0, \mu) = -\frac{y_0 + \mu t_i}{\sigma^2} \qquad \text{(A.12)}$$

$$\begin{aligned}
\frac{\partial}{\partial y_0} S_i(y_0, \mu) &= \frac{1}{\sqrt{\sigma^2 t_i}}\phi\left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}}\right) + \frac{2\mu}{\sigma^2}\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right) \\
&\quad + \frac{1}{\sqrt{\sigma^2 t_i}}\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right)
\end{aligned} \qquad \text{(A.13)}$$

$$\begin{aligned}
\frac{\partial}{\partial \mu} S_i(y_0, \mu) &= \frac{t_i}{\sqrt{\sigma^2 t_i}}\phi\left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}}\right) + \frac{2y_0}{\sigma^2}\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right) \\
&\quad - \frac{t_i}{\sqrt{\sigma^2 t_i}}\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right)
\end{aligned} \qquad \text{(A.14)}$$

Hence

$$\frac{\partial}{\partial y_0} l(y_0, \mu)$$

$$\begin{aligned}
&= \sum_{i=1}^{n} \left(\delta_i\left(\frac{1}{y_0} - \frac{y_0 + \mu t_i}{\sigma^2 t_i}\right) + (1 - \delta_i)\left[\frac{1}{\sqrt{\sigma^2 t_i}}\phi\left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}}\right) + \frac{2\mu}{\sigma^2}\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right) \right. \right. \\
&\quad \left. \left. + \frac{1}{\sqrt{\sigma^2 t_i}}\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right)\right]\left[\Phi\left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}}\right) - \exp(-\frac{2y_0\mu}{\sigma^2}\Phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right)\right]^{-1}\right)
\end{aligned}$$

$$\text{(A.15)}$$

and

$$\frac{\partial}{\partial \mu} l(y_0, \mu)$$

$$= \sum_{i=1}^{n} \left(\delta_i \left(-\frac{y_0 + \mu t_i}{\sigma^2} \right) + (1 - \delta_i) \left[\frac{t_i}{\sqrt{\sigma^2 t_i}} \phi \left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}} \right) + \frac{2y_0}{\sigma^2} \exp \left(-\frac{2y_0 \mu}{\sigma^2} \right) \Phi \left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}} \right) \right. \right.$$

$$\left. \left. -\frac{t_i}{\sqrt{\sigma^2 t_i}} \exp \left(-\frac{2y_0 \mu}{\sigma^2} \right) \phi \left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}} \right) \right] \left[\Phi \left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}} \right) - \exp(-\frac{2y_0 \mu}{\sigma^2} \Phi \left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}} \right) \right]^{-1} \right)$$

$$\text{(A.16)}$$

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. 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.

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.

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.

bovelstad2009 — Bøvelstad, H. M., Nygård, S., and Borgan, Ø. (2009). Survival prediction from clinico-genomic models - a comparative study. *BMC Bioinformatics*, 10(1):413.

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.

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. (1992). *Regression Models and Life-Tables*, pages 527–541. Springer New York, New York, NY.

`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.

`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.

`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.
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).
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.

| tukey |

Tukey, J. (1977). *Exploratory Data Analysis*. Addison-Wesley series in behavioral science. Addison-Wesley Publishing Company.

| 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.

| 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.