Boosting the First-Hitting-Time Regression Model

Vegard Stikbakke

March 12, 2019

Abstract

Empty.

Acknowledgements

Empty.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

sec:intro

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 the proportional hazards model, typically known as Cox regression. 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.

Chapter 2

First-hitting-time models

2.1 Survival data and basic definitions

Survival analysis is the field of studying lifetime and time-to-death data.

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 deaths, and hence the time to the event happens we call time-to-death. The time-to-death 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.

An overview of modelling survival data is Aalen et al. (2008).

2.1.2 Independently censored survival data

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*. We say that we have independent censoring if the censoring indicator d is independent of the time, meaning that

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

An example where we do not have independent censoring is ...

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 $F(t)$ is the cumulative distribution has a derivative $f(t)$ which exists, 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.

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),

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

{eq:cumulative-hazard}

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

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

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

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

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

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

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

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

2.2 Classical inference approaches

2.2.1 Likelihood regression

Consider survival data (t_i, d_i), $i = 1, 2, \ldots, N$, where t_i is the time to death of individual i, and d_i is a censoring indicator of the usual type, meaning it is 1 if the death has been observed, and 0 if not. To make inference on what affects the time to death, 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 number of pieces of information. We gather these in a vector $\mathbf{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,p})$. We may now consider a data set of complete tuples of survival data with covariates,

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

{eq:surv-D}

Now, consider a survival time distribution $\psi(\boldsymbol{\beta})$, 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

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

and a parameterized probability distribution function

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

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 an expression for the probability of observing the observed sample, as a function of the parameters $\boldsymbol{\beta}$ in the distribution. If all censoring indicators are 1, meaning all observations are actual deaths, we are in the usual statistical regression landscape. For each observation, its likelihood is the probability of observing its death at t_i given the parameters and the data,

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

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,

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{N} f(t_i|\boldsymbol{\beta}, x_i). \tag{2.7}$$

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 data $D = \{x_i, y_i\}_{i=1}^n$ is independent and identically distributed from a survival distribution $\psi(\boldsymbol{\beta})$. In the case of an uncensored observation i, meaning $d_i = 1$, it contributes

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

{eq:f}

to the likelihood. If the event has not occurred, the observation is censored, and 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

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

{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

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

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

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

{eq:surv-lik}

It is more convenient to work with the log likelihood,

$$\begin{aligned} l(\boldsymbol{\beta}) &= \log L(\boldsymbol{\beta}) \\ &= \sum_{i=1}^{n} \left[d_i \log f(t_i|\mathbf{x}_i, \boldsymbol{\beta}) + (1 - d_i) \log S(t_i|\mathbf{x}_i, \boldsymbol{\beta}) \right]. \end{aligned} \tag{2.11}$$

{eq:surv-log-lik}

Note that since $A(t) = -\log S(t)$ and $f(t) = \alpha(t)S(t)$, see (2.5) and (2.2), respectively, (2.11) further simplifies to

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

subsec:ph-reg

2.2.2 Proportional hazards regression

In other fields of statistics, we are often most interested in modelling the probability distribution function and the cumulative distribution function. In survival analysis, however, we are typically more interested in modelling the hazard function. The reasons for this will become clear in a minute.

In this subsection, we consider the effect of covariates on the hazard function $\alpha(\cdot)$. How may we use a covariate vector $\mathbf{x}$ in modelling the hazard rate? A very common model to choose here is that of a proportional hazards model, which assumes

$$\alpha(t|\mathbf{x}) = \alpha_0(t) r(\mathbf{x}|\boldsymbol{\beta}), \tag{2.12}$$

{eq:PH}

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

The idea behind the partial likelihood is as follows. Since 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,

$$\frac{\Pr(\text{event happens to individual } i \text{ at time } t)}{\Pr(\text{event happens to any individual } j \text{ at time } t)}. \tag{2.13}$$

{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 $\alpha(t|\mathbf{x}_i)$. 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 (2.13) at all the observed, uncensored, times. Inserting the probabilities into the informal expression in (2.13), an individual with an observed death at t_i contribues to the likelihood with

$$\frac{\alpha_0(t_i) r(\mathbf{x}_i)}{\sum_{j \in R(t_i)} \alpha_0(t_i) r(\mathbf{x}_j)}. \tag{2.14}$$

{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 (2.14), is

$$\mathrm{pl}(\boldsymbol{\beta}) = \prod_{i:\ d_i=1} \frac{\alpha_0(t_i) r(\mathbf{x}_i)}{\sum_{j \in R(t_i)} \alpha_0(t_i) r(\mathbf{x}_j)}, \tag{2.15}$$

{eq:pl}

The baseline hazard will cancel out in expression (2.15), and we are left with just the relative risk functions,

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

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 fully model the effect of covariates without having to consider the baseline hazard.

We have not said anything about what $r(\mathbf{x})$ looks like. The most common choice, by far, for $r(\mathbf{x})$ is

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

which leads to the famous Cox model (Cox and Miller, 1965). The Cox model is an attractive model because the effect of a unit increase in an element of $\boldsymbol{\beta}$ has a nice interpretation. Assume we have two observations with covariate vectors $\mathbf{x}_1$ and $\mathbf{x}_2$, and that $\mathbf{x}_2$ is equal to $\mathbf{x}_1$ except for in element j, where $x_{2j} = x_{1j} + 1$. Then the ratio of the two hazard rates becomes

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

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

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

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. We will here give a simple example to illustrate its use.

2.2.3 Cox regression example

Lorem ipsum.

2.2.4 Issues with the proportional hazards assumption

Assuming (2.12), i.e. $\alpha(t|\mathbf{x}) = \alpha_0(t) r(x|\boldsymbol{\beta})$, 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 $\alpha(t|\mathbf{x})$ 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. There exist alternative models, which do not assume 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

$$\alpha(t|\mathbf{x}) = \beta_0(t) + \beta_1(t)x_1(t) + \beta_2(t)x_2(t) + \ldots + \beta_p(t)x_p(t), \tag{2.16}$$

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 this model, which has not seen methods developed for it so as to be able to use it in typical clinical settings for biomedical data, such as on gene expression data.

2.2.5 Robustness of Cox when the PH assumption is violated

Although the PH assumption is often not valid, in practice, Cox regression tends to work well.

Need to find a citation here.

2.3 First hitting time models

sec:FHT

So far we have done regression by modelling the hazard rate $\alpha(\cdot)$. Therefore we have not thought much about how a time-to-death 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. One way to think about how these times are generated, without thinking about probability distributions just yet, is to imagine that each individual has an underlying stochastic process, a health process $Y(t)$, say. When this health process reaches some barrier, or some end state, the individual dies. This is a conceptual appealing framework, because, instead of just a stochastic lifetime, it introduces the concept of distance to death. We may call the time the health process hits this barrier or end state for the *first hitting time* (FHT) of a boundary or threshold state, by sample paths of a stochastic health process. This health process may be latent, i.e., unobservable, or it may be observable. 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. FHT models have been used to describe the length of a hospital stay (Whitmore, 1975; Eaton and Whitmore, 1977), and to model the duration of a strike (Lancaster, 1972). They have also been used to estimate degradation in components (Whitmore, 1995), and to assess lung cancer risk in railroad workers (Lee et al., 2004).

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. In some applications, however, it varies with time, i.e., $\mathcal{B}(t)$. This is, however, out of the scope of this thesis.

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 S defined as

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

{eq:fht-t}

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

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

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

This condition is sometimes plausible and a desirable model feature. This will become apparent when we deal with regression.

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 already referring to a well established, and quite different, topic in econometrics. First-hitting-time models have been applied to many different fields, including medicine, engineering, and economics, and used, for example to describe the survival time of a transplant patient, the duration time of a strike (Lancaster, 1972), the failure time of an engineering system , and so on.

citation needed

citation needed

2.3.2 Examples of first-hitting-time models

The parent stochastic process may take many forms, from Wiener processes to Markov chains. Similarly, the boundary state may take various forms. For example, it may be a fixed threshold in a Wiener process or an absorbing state

in a Markov chain. This choice in process and boundary shows that the FHT framework is highly flexible. The choice of boundary must fit the process, such that the boundary set is a subset of the state space.

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

$$P(T > t) = P(Z(t) < y_0). \tag{2.19}$$

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.

If we consider the health process to be analogous with a person's health, this makes sense. 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 do, however, want a monotonic restriction on the movement of the health process, we may use a gamma process (Lee and Whitmore, 2006). This might make sense if the health process is meant to model e.g. the breakdown of a structure. Another 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

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

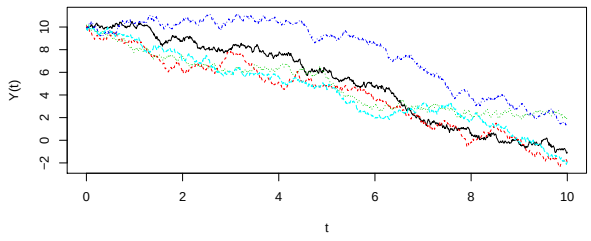
$$Y(t) = y_0 - \mu t + \sigma W(t), \tag{2.20}$$

{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).

plot:wiener

Figure 2.1: Example of 5 Wiener process paths with initial value $y_0 = 10$ and drift $\mu = 1$.



Clearly, for a Wiener process starting in $y_0 > 0$ and with a downwards drift, i.e. $\mu > 0$, the movement is markedly in the direction of zero. If σ^2 is small in comparison to the drift, as well as that the initial level is sufficiently high in comparison to σ^2, then the process will move in almost a straight line, such that

$$X(t) \approx y_0 - \mu t.$$

Consequently, the first hitting time will be nearly a deterministic function of y_0 and μ,

$$T \approx \frac{y_0}{\mu}. \tag{2.21}$$

This is quite visible in Figure 2.3.3, which shows examples of 5 Wiener process paths with initial value $y_0 = 10$ and drift $\mu = 1$. If σ^2 is relatively large compared to the drift, the diffusion part is more dominant and thus the hitting time is less predictable (Aalen et al., 2008).

2.3.4 Inverse Gaussian distribution and different parameterizations

2.3.5 Wiener process and Inverse Gaussian first hitting time

If we choose the stochastic process to be a Wiener process like in (2.20), and we let the boundary be the non-positive numbers, $\mathcal{B} = (-\infty, 0]$, then (2.17) becomes

$$T = \min_t \left(t \colon Y(t) \in \mathcal{B}\right), \tag{2.22}$$

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 the Wiener process follows an inverse Gaussian distribution (Chhikara, 1988), with probability distribution function (pdf)

$$f(t|y_0, \mu, \sigma^2) = \frac{y_0}{\sqrt{2\pi\sigma^2 t^3}} \exp\left[-\frac{(y_0 + \mu t)^2}{2\sigma^2 t}\right], \tag{2.23}$$

{eq:ig-pdf}

and cumulative distribution function (cdf)

$$F(t|\mu, \sigma^2, y_0) = \Phi\left[-\frac{\mu t + y_0}{\sqrt{\sigma^2 t}}\right] + \exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left[\frac{\mu t - y_0}{\sqrt{\sigma^2 t}}\right]. \tag{2.24}$$

{eq:ig-cdf}

As previously discussed in subsection 2.3.1, note that if the drift μ is positive, then it is not certain that the process will ever reach 0. Hence the probability distribution function in (2.23) is improper. In this case, the probability of the time not being finite is

$$\Pr(T = \infty) = 1 - \Pr(T < \infty) = 1 - \exp(-2y_0\mu),$$

see Cox and Miller (1965). Since we in survival analysis prefer working with the survival function $S(t) = 1 - F(t)$ rather than the cdf $F(t)$, we note that $S(t)$ becomes

$$S(t|\mu, \sigma^2, y_0) = \Phi\left[\frac{\mu t + y_0}{\sqrt{\sigma^2 t}}\right] - \exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left[\frac{\mu t - y_0}{\sqrt{\sigma^2 t}}\right], \tag{2.25}$$

{eq:ig-surv}

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

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

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

$$\phi(x) = \frac{\exp\left(-x^2/2\right)}{\sqrt{2\pi}}. \tag{2.26}$$

Note that we in (2.25) used the fact that the cdf of the standard normal distribution is symmetric around 0, meaning that we were able to swap

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

for

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

2.3.6 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 (2.23) and the survival function $S(t|\mu, \sigma^2, y_0)$ in (2.25) only depend on these parameters through the ratios μ/σ and y_0/σ. Hence, there are in reality 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). We will use this to make inference.

2.3.7 The shape of the hazard rate

The hazard rate is $\alpha(t) = f(t)/S(t)$ (2.2). Regardless of the initial value, this converges to the same limiting hazard. If y_0 is close to zero, we essentially get a decreasing hazard rate. If y_0 is far from zero, this gives an essentially increasing hazard rate. If y_0 is somewhat inbetween, we get a hazard rate which first increases and then decreases (Aalen et al., 2008).

This is unclear

Add plots of hazard rates here. see page 401 in ABG

2.3.8 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).

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. The hazard function converges to

$$\lim_{t\to\infty} \alpha(t) = \frac{1}{2}\left(\frac{\mu}{\sigma}\right)^2 = 0.5\mu^2 \tag{2.27}$$

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

2.3.9 Regression

subsec:IG-reg

We may introduce effects from covariates by allowing μ and y_0 to depend on covariates $\mathbf{x}$ and $\mathbf{z}$. 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 it must be positive in our framework,

$$\mu(\boldsymbol{\beta}) = \boldsymbol{\beta}^{\mathrm{T}}\mathbf{x} = \sum_{j=1}^{p} \beta_j x_j, \tag{2.28}$$

{eq:y0}

$$y_0(\boldsymbol{\gamma}) = \exp(\boldsymbol{\gamma}^{\mathrm{T}}\mathbf{z}) \Rightarrow \ln y_0(\boldsymbol{\gamma}) = \boldsymbol{\gamma}^{\mathrm{T}}\mathbf{z} = \sum_{j=1}^{d} \gamma_j z_j. \tag{2.29}$$

{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 $\mathbf{x}$ and $\mathbf{z}$ share none, some, or all elements. We will discuss consequences of this later.

Inserting the pdf (2.23) and the survival function (2.25) into the log-likelihood (2.10), we get that the log-likelihood of a survival data set with the inverse gaussian FHT model, i.e.,

$$\begin{aligned} l(y_0, \mu, \sigma) = \sum_{i=1}^{n} d_i \Bigg(& \ln y_0 - \frac{1}{2}\ln\left(2\pi\sigma^2 t_i^3\right) - \frac{(y_0 + \mu t_i)^2}{2\sigma^2 t_i} \Bigg) \\ & + (1 - d_i)\ln\Bigg(\Phi\left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}}\right) - \exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right)\Bigg). \end{aligned} \tag{2.30}$$

{eq:loglik}

2.3.10 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 a *regularized* model at the moment, nor to do automatic variable selection. **This is the reason for my thesis.**

Add regression example

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, means one which has a low signal-to-noise ratio, and which in general performs poorly. For classification purposes it is easy to give a good example: A weak learner is one which performs only slightly better than random uniform chance. In the binary classification setting, then, it would only perform slightly better than a coin flip. For regression, a weak learner is for example a linear least squares model of only one variable, and having only a small parameter effect for that variable. Meanwhile, a strong learner should be able to perform in a near-perfect fashion, for example attaining high accuracy on a prediction task. I will first attend to 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, 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 $(\mathbf{x}_i, y_i)_{i=1,\ldots,n}$, where $x_i \in \mathbb{R}^p$ 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 $\mathbf{x}_{\text{new}}$ 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 $\mathbf{x}$, 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. 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. Finally, based on the misclassification rate of this classifier, calculate a weight α to give to this classifier. Currently, the classifier is $F_1(\cdot) = \alpha_1 h_1(\cdot)$. In the next iteration, aaply 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).

Overfitting occurs when the out-of-sample error starts to increase. At this point, the model is starting to be too sensitive to the structure of the specific data set it is estimated on. One way of thinking about it is that it is starting to fit to the error terms. Since what we actually care about is the performance on a test set, we want to stop just before the model starts overfitting.

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.

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.

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

$$D = \{\mathbf{X}^{(i)}, \mathbf{Y}^{(i)}\}_{i=1,\ldots,n} \tag{3.1}$$

containing the values of an outcome variable $\mathbf{Y}$ and predictor variables $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_p$, forming covariate matrix $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_p)$. We assume that the samples $i = 1, \ldots, n$ are generated independently from an identical distribution over the joint space $\mathcal{X} \times \mathcal{Y}$. The input space of $\mathbf{X}$ 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 $\mathbf{Y}$ and $\mathbf{X}$ and to obtain an "optimal" prediction of $\mathbf{Y}$ given $\mathbf{X}$. 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 $\mathbf{Y}|\mathbf{X} = \mathbf{x}$ follows some probability distribution function (pdf)

$$\psi(\mathbf{Y}|\theta(\mathbf{X} = \mathbf{x})), \tag{3.2}$$ {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. Further, we will model θ as a functional of the covariates $\mathbf{X}$, with conditional expectation given the observed value $\mathbf{X} = \mathbf{x}$ as

$$g(\mathbb{E}(\theta(\mathbf{X} = \mathbf{x})) = f(\mathbf{x}), \tag{3.3}$$

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

$$\mathbb{E}(\theta(\mathbf{X} = \mathbf{x})) = g^{-1}(f(\mathbf{x})). \tag{3.4}$$

This means that the conditional expectation of θ given the observed $\mathbf{x}$ is a transformation of the additive predictor $f(\mathbf{x})$ 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

$$\mathbb{E}(\theta(\mathbf{X} = \mathbf{x})|\mathbf{X} = \mathbf{x}) = \exp(f(\mathbf{x})), \tag{3.5}$$

which will constrain the expectation to be a positive number. Further, the additive predictor $f(\cdot)$ consists of the additive effects of the single predictors,

$$f(\mathbf{x}) = \beta_0 + f_1(x_1) + \ldots + f_p(x_p), \tag{3.6}$$ {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.

add citations on base learners here?

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 3.7 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 $\mathbf{y}$ 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. Since we maximize the log-likelihood in statistics, very often, the loss function ρ is derived from the negative log likelihood of the distribution of the response (Mayr et al., 2014a; Bøvelstad and Borgan, 2011), which we denoted ψ in (3.2). In these cases, the loss function, which works on one set of observations $(\mathbf{x}_i, \mathbf{y}_i)$, is

$$\rho(\mathbf{y}, \theta(\mathbf{x})) = -\log \psi(\mathbf{y}|\theta(\mathbf{x}), \tag{3.7}$$

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.

Drop this? Or add citation?

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 $D = \{\mathbf{x}_i, y_i\}_{i=1}^N$ 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,

$$g(\mathbf{x}) = \mathbf{x}. \tag{3.8}$$

Further, it means that

$$\mathbb{E}(\mu(\mathbf{X} = \mathbf{x})) = f(\mathbf{x}). \tag{3.9}$$

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

$$.f(y|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp - \frac{(y-\mu)^2}{2\sigma^2}, \tag{3.10}$$

and we derive the loss function ρ accordingly, yielding

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

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 Optimization

Obtaining the optimal prediction is accomplished by minimizing ρ with regard to the additive predictor $f(\cdot)$. We have only observed a data set D, but we wish to generalize the results to unseen data sets of a similar type. Therefore, what is typically done is to minimizes the loss over an unseen "hold-out" sample, often called the test error. Other names are generalization error and out-of-sample error. For a specific data set D, we can calculate the loss function on each observation, and sum these losses. This is what is often called the empirical risk of the loss ρ over the data set D, which we will denote R_D,

$$R_D(\theta) = \sum_{i=1}^{n} \rho(y_i, \theta(x_i)), \tag{3.11}$$

{eq:empirical-risk}

where again the parameter θ is a functional

$$\theta(x) = g(f(x)), \tag{3.12}$$

$g(\cdot)$ is the chosen link function, and $f(\mathbf{x})$ is the additive predictor. Other names for $R_D(\cdot)$ are in-sample error and training error.

This section needs to be rewritten. Maybe the above is good???

Since D arises from a data distribution, R_D is a realization of a more general loss value. We wish to learn about the general structure of D, and as such are we most interested in the expected loss, also known as the generalization or test error,

$$\mathrm{Err}_D = \mathrm{E}[\rho(Y, f(\mathbf{X}))|D],$$

where (X, Y) is drawn randomly from their joint distribution and the training set D is held fixed. It is infeasible to do effectively in practice and hence we must instead estimate the expected prediction error,

$$\mathrm{Err} = \mathrm{E}[\mathrm{Err}_D] = \mathrm{E}_D\left[\rho(Y, f(\mathbf{X}))|D\right], \tag{3.13}$$

{eq:err}

i.e., average over many different test sets. As mentioned, in practice we observe a sample data set D. For this sample, we can calculate the training error – the empirical risk. To estimate Err (3.13), one can do two things. First, if the observed sample is large enough, one can choose a portion of this, say 20%, to be used as a hold-out test set. Call this data set for D_{test} We then estimate our model and its parameters based on the other 80%, and make an estimate of the generalization error Err by seeing how our estimated model performs on the hold-out test set. We call the resulting error

$$\widehat{\mathrm{Err}}_{\text{test}} = \frac{1}{M} \sum_{i=1}^{M} \rho(y_i, \hat{f}(\mathbf{x}_i)),$$

for test error. If the observed sample is not large enough, one can calculate a K-fold cross-validated test error. In this case, one divides the data set into

K parts (folds), and for each fold, one lets it be the hold-out data set, and estimate a model using only the other $K-1$ folds. In this way, one gets K test errors, and so the cross-validated test error is the mean of these. See later for a more detailed description.

This section needs to be rewritten.

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 $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ exist, then the gradient of G at $\mathbf{x}$ is the vector of all its partial derivatives at $\mathbf{x}$, namely

$$\nabla G(\mathbf{x}) = \left(\frac{\partial G(\mathbf{x})}{\partial x_1}, \frac{\partial G(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial G(\mathbf{x})}{\partial x_m} \right). \tag{3.14}$$

The motivation behind the gradient descent algorithm is that in a small interval around a point $\mathbf{x}_0 \in \mathbb{R}^m$, 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 $\mathbf{x}^{[0]}$ to a new value $\mathbf{x}^{[1]}$, we end up with a slightly lower function value: The new function value $G(\mathbf{x}^{[1]})$ will be less than $G(\mathbf{x}^{[0]})$. 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, 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 Gradient boosting

In a seminal paper, Friedman (2001) developed an iterative algorithm for fitting an additive model (3.6) called gradient boosting. He showed that AdaBoost performs this algorithm for a particular exponential loss function, which 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 AdaBoost argument. This provided a way of viewing boosting through

algo:grad-desc

Algorithm 1 Gradient descent

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

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

grad-desc-iter

2. Increase m by 1.
3. Calculate the direction to step in, $\mathbf{g}_{m-1} = -\nabla G(\mathbf{x}^{[m-1]})$.
4. Solve the line search to find the best step length $a^{[m]}$,

$$a^{[m]} = \underset{a}{\operatorname{argmin}}\, \mathbf{x}^{[m-1]} + a \cdot \mathbf{g}_{m-1}.$$

5. The step in iteration m becomes $\mathbf{h}_m = a^{[m]} \cdot \mathbf{g}_{m-1}$.
6. Let $\mathbf{x}^{[m]} = \mathbf{x}[m-1] + \mathbf{h}_{m-1}$.
7. If $m < m_{\text{stop}}$, go to step (2).
8. The resulting minimum point is $\mathbf{x}^{[m_{\text{stop}}]} = \mathbf{x}^{[0]} + \sum_{m=1}^{M} \mathbf{h}_m(\mathbf{x}^{[m]})$.

a statistical lens, and connected the successful machine learning approach to the world of statistical modelling. Gradient boosting does gradient descent in parameter space. Hence gradient boosting is gradient descent in the functional parameter space spanned by the base learners (Friedman, 2001). Boosting can be viewed as an optimization procedure in functional space.

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

$$\hat{\eta} = \underset{\eta}{\operatorname{argmin}} \left(\mathbb{E}_{Y,X}[\rho\{Y, \eta(X)\}] \right), \tag{3.15}$$

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

$$R_D(\eta) = \frac{1}{n} \sum_{i=1}^{n} \rho(y_i, \eta(\mathbf{x}_i)). \tag{3.16}$$

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

The key idea of statistical boosting is to iteratively fit the different predictors with simple regression functions (base-learners) and combine the estimates to a predictor. In case of gradient boosting, the base-learners are fitted to the negative gradient of the loss function; this procedure can be described as gradient descent in function space (Bühlmann and Hothorn, 2007).

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

$$\eta^* = \underset{\eta}{\operatorname{argmin}}\, R_D(\eta) = \underset{\eta}{\operatorname{argmin}} \sum_{i=1}^{n} \rho(\mathbf{y}_i, \eta(\mathbf{x}_i)), \tag{3.17}$$

{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

$$\hat{\eta} = \beta_0 + \sum_{m=1}^{m_{\text{stop}}} f^{[m]}, \tag{3.18}$$

where the first term, β_0, is an initial guess, effectively $f^{[0]}$, 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 $\mathbf{x}$, this is

$$-\frac{\partial}{\partial \eta}\rho(y, \eta(x)), \tag{3.19}$$

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

$$\mathbf{u} = (u_i)_{i=1}^{N} = \left(-\frac{\partial}{\partial \eta}\rho(y_i, \hat{\eta})\big|_{\hat{\eta}=\hat{\eta}^{[m-1]}(x_i)}\right)_{i=1}^{N}, \tag{3.20}$$

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 $\mathbf{x}_i$. 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 $i = 1, 2, \ldots, n$.

However, while this nonparametric approach would reduce the error of each data point, it will not generalize. 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 (3.33), over all values of $\mathbf{X}$ and Y in the joint distribution. Additionally, we wish to have an interpretable model. Therefore, we must impose smoothness to neighboring points in the $\mathcal{X}$ space. We can do this by choosing steps of *functions* instead of function *values*. Therefore, since the solutions are parameterized, the approach by Friedman (2001) develops a *functional* gradient descent (FGD) algorithm. To perform a parameterized step, we choose a so-called base learner

$$h(\cdot). \tag{3.21}$$

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. Secondly, 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,

$$\mathbf{u}^{[m-1]} = \left(-\frac{\partial}{\partial \eta} \rho(y_i, \eta) \big|_{\hat{\eta}=\hat{\eta}^{[m-1]}(x_i)} \right)_{i=1}^{N}, \tag{3.22}$$

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, the step which has the steepest gradient is the member of the base learner class which produces the function $\hat{h}^{[m]}$ *most parallel* to $\mathbf{u}^{[m-1]}$. Another way of looking at it is that it this is the h most correlated with $\mathbf{u}^{[m-1]}$ over the data distribution. This means that this $\hat{h}^{[m]}$ is an approximation of the generalized residuals $\mathbf{u}^{[m-1]}$, 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 ordinary least squares, then this will be $\hat{h}^{[m]} = \left(\mathbf{u}^{[m-1]}\right)^T \hat{\boldsymbol{\beta}}^{[m]}$, where

$$\hat{\boldsymbol{\beta}}^{[m]} = \left(\left(\mathbf{u}^{[m-1]}\right)^T \mathbf{u}^{[m-1]} \right)^{-1} \left(\mathbf{u}^{[m-1]}\right)^T \mathbf{y}. \tag{3.23}$$

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

$$a^{[m]} = \underset{a}{\operatorname{argmin}} R_D \left(\hat{f}^{[m-1]} + a^{[m]} \cdot \hat{h}^{[m]}. \right). \tag{3.24}$$

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

$$\hat{f}^{[m]}(\cdot) \leftarrow \hat{f}^{[m-1]}(\cdot) + a^{[m]} \hat{h}^{[m]}(\cdot). \tag{3.25}$$

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

3.4.1 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

algo:fgd

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

$$\hat{f} = \underset{f}{\operatorname{argmin}} R(f) = \underset{f}{\operatorname{argmin}} \sum_{i=1}^{n} \rho(y_i, f(x_i). \tag{3.26}$$

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.,

$$\hat{f}_0(\cdot) = \beta_0(\cdot) = \underset{c}{\operatorname{argmin}} R(c). \tag{3.27}$$

3. Specify a base learner h.

4. Increase m by 1.

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

$$\mathbf{u}^{[m-1]} = \left(-\frac{\partial}{\partial f} \rho(y_i, f(x_i)) \big|_{f=\hat{f}^{[m-1]}} \right)_{i=1}^{N} \tag{3.28}$$

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

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

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

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

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

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

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) find citation?

As we will see, most modern boosting algorithms omit the step of the line search to find a_m, but instead always uses a 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., with sufficient shrinkage, but the convention is to use $\nu = 0.1$ (Mayr et al., 2014a). This reduces the complexity of the algorithm, and makes the number of parameters to estimate lower. 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.

subsec: iterations

3.4.2 Number of 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. 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.

3.4.3 Practical considerations

When boosting, one must (or should) center and scale the matrix X.

3.5 likelihood-based boosting

Lorem ipsum. (De Bin, 2016) (Tutz and Binder, 2006).

3.6 L_2Boost

With the generic functional gradient boosting algorithm (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. This gives great flexibility.

In the original paper (Friedman, 2001), he derived such 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 of 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 $\mathbf{u}$ then becomes simply the residual vector,

$$\frac{\partial \mathrm{L}(y, f(\mathbf{x}))}{\partial x_i} = (y - f(x_i)), \quad i = 1, \ldots, n,$$

and hence the boosting steps become 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 $\nu_m = \nu = 1$ for all iterations $m = 1, \ldots, M$. They also prove some nice important theoretical results for L2Boost.

more on L2Boost!!

3.6.1 L_2Boost example

Lorem ipsum.

3.7 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 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 $\mathbf{X}$ consists of gene expressions, it is obvious that the response variable y is not dependent on every single gene.

3.7.1 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

algo:L2

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,

$$\hat{f} = \underset{f}{\operatorname{argmin}}\, R(f) = \underset{f}{\operatorname{argmin}} \sum_{i=1}^{n} y_i - \hat{f}(x_i). \tag{3.29}$$

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

$$f_0(\cdot) = \underset{c}{\operatorname{argmin}}\, R(c). \tag{3.30}$$

3. Let the base learner class h be the least squares model, i.e.,

$$h(\mathbf{x}, \boldsymbol{\beta}) = \mathbf{x}^T \boldsymbol{\beta} = \sum_{j=1}^{p} \beta_j x_j \tag{3.31}$$

4. Increase m by 1.

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

$$\mathbf{u}^{[m-1]} = \left(y_i - \hat{f}(x_i)\right)_{i=1}^{N} \tag{3.32}$$

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

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

   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 $\hat{f}(\cdot) = \hat{f}_M(\cdot) = \sum_{m=0}^{M} f_m(\cdot)$.

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 is to add only 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 (3.6), which minimizes the empirical risk of a chosen loss function on a data set $D = \{x_i, y_i\}_{i=1}^N$,

$$\eta^* = \underset{\eta}{\operatorname{argmin}} R_D(\eta) = \underset{\eta}{\operatorname{argmin}} \sum_{i=1}^{n} \rho(\mathbf{y}_i, \eta(\mathbf{x}_i)), \tag{3.33}$$

{eq:argmin-eta}

where the parameter η is an additive predictor

$$\eta(\mathbf{x}) = g(\theta) = \beta_0 + f(\mathbf{x}) = \beta_0 + \sum_{j=1}^{p} f_j(x_j), \tag{3.34}$$

and $g(\cdot)$ is a known, monotonic link function chosen appropriately. 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 (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 have the same structure, but each uses its own covariate. For example, if we use a linear least squares model as base learners, the set of base learners would be

$$\mathcal{H} = \{h_1(\mathbf{x}; \beta_1) = \beta_1 x_1, h_2(\mathbf{x}; \beta_2) = \beta_2 x_2, \ldots, h_p(\mathbf{x}; \beta_p) = \beta_p x_p\}. \tag{3.35}$$

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,

$$\mathbf{u}^{[m-1]} = \left(-\frac{\partial}{\partial \eta} \rho(y_i, \eta) \big|_{\eta = \hat{\eta}^{[m-1]}(x_i)} \right)_{i=1}^{N}. \tag{3.36}$$

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

$$\hat{h}_1^{[m]}(\cdot),\ \hat{h}_2^{[m]}(\cdot),\ \ldots,\ \hat{h}_p^{[m]}(\cdot), \tag{3.37}$$

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

$$j^{[m]} = \underset{j\in\{1,2,\ldots,p\}}{\operatorname{argmin}} \sum_{i=1}^{N} \left(u_i - \hat{h}_j^{[m]}\right)^2. \tag{3.38}$$

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

$$\hat{f}^{[m]}(\cdot) \leftarrow \hat{f}^{[m-1]}(\cdot) + \nu \cdot \hat{h}_{j^{[m]}}. \tag{3.39}$$

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

$$\hat{f}_{j^{[m]}}^{[m]}(\cdot) \leftarrow \hat{f}_{j^{[m]}}^{[m-1]}(\cdot) + \nu \cdot \hat{h}_{j^{[m]}}, \tag{3.40}$$

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

$$\hat{f}_j^{[m]}(\cdot) \leftarrow \hat{f}_j^{[m-1]}. \tag{3.41}$$

We continue iterating until the iteration number m reaches the pre-specified stopping iteration m_{stop}. The final additive predictor becomes

$$\hat{\eta} = \hat{\eta}^{[m_{\text{stop}}]} = \beta_0 + \sum_{m=1}^{m_{\text{stop}}} \nu \cdot \hat{h}_{j^{[m]}}^{[m]}(\cdot). \tag{3.42}$$

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

$$\hat{\eta}(\mathbf{x}) = \beta_0 + \sum_{j=1}^{p} \hat{f}_j(x_j). \tag{3.43}$$

For a schematic overview of the algorithm, see Algorithm 4.

algo:component-wise

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,

$$\hat{f} = \underset{f}{\operatorname{argmin}} R(f). \tag{3.44}$$

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.,

$$\beta_0 = \underset{c}{\operatorname{argmin}} R(c). \tag{3.45}$$

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 $\mathbf{X}$.

first-step

4. Increase m by 1.

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

$$\mathbf{u}^{[m-1]} = \left(-\frac{\partial}{\partial f} \rho(y_i, f(x_i)) \big|_{f=\hat{f}^{[m-1]}} \right)_{i=1}^N. \tag{3.46}$$

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

$$\hat{h}_1^{[m]}(\cdot), \hat{h}_2^{[m]}(\cdot), \ldots, \hat{h}_p^{[m]}(\cdot). \tag{3.47}$$

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

$$j^{[m]} = \underset{j \in \{1,2,\ldots,p\}}{\operatorname{argmin}} \sum_{i=1}^N \left(u_i - \hat{h}_j^{[m]} \right)^2. \tag{3.48}$$

last-step

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

$$\hat{f}^{[m]}(\cdot) \leftarrow \hat{f}^{[m-1]}(\cdot) + \nu \cdot \hat{h}_{j^{[m]}}^{[m]}(\cdot). \tag{3.49}$$

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

10. Return the final boosted additive predictor

$$\hat{f}(\cdot) = \hat{f}^{[m_{\text{stop}}]}(\cdot) = \beta_0 + \sum_{m=1}^{m_{\text{stop}}} \nu \cdot \hat{h}_{j^{[m]}}^{[m]}(\cdot) \tag{3.50}$$

3.8 Boosting performs data-driven variable selection

If the number of iterations m_{stop} is not very large compared to the number of variables, the component-wise gradient boosting algorithm will carry out automatic variable selection. What this means is that based on their explanatory power, the base learners applied to irrelevant variables will never be added into the model, and therefore many of the columns of $\mathbf{X}$ 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 than once. This is because some predictors are more correlated with the output than others. Therefore some components will lead to better improvements and those corresponding base learners will thus be more frequently selected. Therefore the component-wise boosting algorithm has inherent variable selection.

3.9 Selecting m_{stop}

As we mentioned in subsection 3.4.2, 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., maximizing 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 model's degrees of freedom. Methods by Chang et al. (2010) try this. This is problematic for several reasons. For L_2Boost, Bühlmann and Hothorn (2007) suggest that $\text{df}(m) = \text{trace}(B_m)$ is a good approximation. Here B_m is the hat matrix resulting from the boosting algorithm. This was, however, shown by Hastie (2007) to always underestimate the actual degrees of freedom. Mayr et al. (2012b) propose a sequential stopping rule using subsampling. However this is computationally very expensive and not really used in practice. 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.Cross-validation is flexible and easy to implement. It is somewhat computationally demanding, because it requires several full runs of the boosting algorithm.

citation?

3.9.1 Other selection methods

The number of iterations in the boosting procedure, M, is a tuning parameter. It acts as a regularizer. AIC, etc.

subsec:K-fold

3.9.2 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. We will use it to approximate the prediction error. In cross-validation, the data is split randomly 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

$$\text{CV}(m) = \sum_{k=1}^{K} \sum_{i \in \kappa(k)} \rho(y_i, \hat{y}_i^{-\kappa(k)}). \tag{3.51}$$

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

$$m_{\text{stop}} = \underset{m}{\text{argmin}}\, \text{CV}(m). \tag{3.52}$$

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.

3.9.3 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 censored 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).

3.9.4 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 true for real-life survival data, especially. In typical survival time data sets one typically has a small effective sample size (number of observed events). We can easily imagine that for two different splits of the data, we can end up with quite different values for m_{stop}. 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). Seibold et al. (2018) suggest simply repeating the cross-validation scheme. They show that repeating even 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,

$$\text{RCV}(m) = \sum_{j=1}^{J} \sum_{k=1}^{K} \sum_{i \in \kappa(j,k)} \rho(y_i, \hat{y}_i^{-\kappa(j,k)}). \tag{3.53}$$

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

$$m_{\text{stop}} = \underset{m}{\text{argmin}}\, \text{RCV}(m). \tag{3.54}$$

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.

3.10 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. 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 modelling technique that relates not only the mean, but all parameters of the outcome distribution to the available covariates. GAMLSS are 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).

3.11 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 $\mathbf{y}_i$ 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

$$f_{\text{dens}}(y_i|\boldsymbol{\theta}_i), \tag{3.55}$$

{gamlss-density}

may depend on K distribution parameters

$$\boldsymbol{\theta}_i = (\theta_{i,1}, \theta_{i,2}, \ldots, \theta_{i,K})^T. \tag{3.56}$$

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

$$g_k(\cdot). \tag{3.57}$$

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

$$x_{k,1}, x_{k,2}, \ldots, x_{k,p_k} \tag{3.58}$$

are the covariates in the submodel of parameters θ_k. A GAMLSS is given by the equations

$$\eta_k \coloneqq g_k(\theta_k) = \beta_{k,0} + \sum_{j=1}^{p_k} f_{k,j}(x_{k,j}), \tag{3.59}$$

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

$$f_{k,j}(x_{k,j}) = x_{k,j}\beta_{k,j}, \tag{3.60}$$

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

$$\boldsymbol{\theta}_i = \theta = \mu, \tag{3.61}$$

i.e., the conditional mean.

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

$$\log\{f_{\text{dens}}(y|\boldsymbol{\theta})\}, \tag{3.62}$$

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 $\mathbf{x}$. Hence the total log-likelihood is

$$l(\boldsymbol{\theta}) = \sum_{i=1}^{n} \log \left(f_{\text{dens}}(y_i|\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}(x_i)} \right), \tag{3.63}$$

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,

$$\hat{\theta}_k = g_k^{-1}(\hat{\eta}_{\theta_k}), \tag{3.64}$$

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 was developed, called *gamboostLSS* (Mayr et al., 2012a).

gamboostLSS uses a strategy for multidimensional boosting proposed by Schmid et al. (2010). Thomas et al. (2018) later coined the term "cyclical" to describe this approach. We will also use this term to describe this algorithm. The main idea of the cyclical multidimensional boosting algorithm is to have a boosting step for each parameter k, in each iteration, and to successively update the predictors in each iteration, using the estimates of the other distribution parameters as offset values. We cycle through all parameter dimensions in each boosting iteration. The *gamboostLSS* algorithm uses this strategy.

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 subsection XXX. This approach is also much more commonly used, since it has been shown that component-wise boosting algorithms often are powerful.

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 prediction. Since we have a prediction function which uses a vector of distribution parameters, we must take the partial derivatives. These negative partial derivatives are

$$-\frac{\partial}{\partial \eta_k} \rho(y, \boldsymbol{\eta}) = \frac{\partial}{\partial \eta_k} \log(f_{\text{dens}}(y|\boldsymbol{\theta})), \tag{3.65}$$

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 $\mathbf{u}_k$ $k = 1, 2 \ldots, K$. 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

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

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

$$h_{1,1}, \ldots, h_{1,p_1}, h_{2,1}, \ldots, h_{2,p_2}, \ldots, h_{K,p_K}. \tag{3.66}$$

We use the base learners to estimate a predictor to add into the model, based on the covariates and the residuals.

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

$$\hat{\eta}_k^{[0]} = \beta_{k,0}, \tag{3.67}$$

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

In iteration m, after having cycled through to component k, the estimated vector of additive predictors is

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

meaning all parameter dimensions $1, 2, \ldots, k-1$ have been updated in the current iteration m. The following dimensions $k+1, \ldots, K$ have not, and we are now going to update dimension k. To make clear the fact that we are in the middle of iteration m, and have so far updated the first $k-1$ dimensions, we denote the vector

$$\hat{\boldsymbol{\eta}}_{k-1}^{[m]}. \tag{3.68}$$

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

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

Again, like in a regular component-wise boosting algorithm, we fit all component-wise base learners separately to this residual vector $\mathbf{u}_k^{[m-1]}$. Of these learners,

select the best fitting component $j_k^{[m]}$ like previously, by selecting the estimated learner which fits best according to RSS,

$$j_k^{[m]} = \underset{j \in \{1,2,\ldots,p_k\}}{\operatorname{argmin}} \sum_{i=1}^{N} \left(u_{k,i}^{[m-1]} - \hat{h}_{k,j}^{[m]} \right)^2 . \tag{3.69}$$

We update the additive predictor in dimension k by the usual

$$\hat{f}_k^{[m]} \leftarrow \hat{f}_k^{[m-1]} + \nu \cdot \hat{h}_{j_k^{[m]}}^{[m]}(\cdot). \tag{3.70}$$

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

$$\begin{aligned}
&\frac{\partial}{\theta_1}\rho\left(y, \eta_1^{[m-1]}, \eta_2^{[m-1]}, \eta_3^{[m-1]}, \ldots, \eta_{K-1}^{[m-1]}, \eta_K^{[m-1]}\right) \xrightarrow{\text{calculate}} \mathbf{u}_1^{[m-1]} \xrightarrow{\text{fit and update}} \hat{f}_1^{[m]} \\
&\quad \frac{\partial}{\eta_2}\rho\left(y, \hat{\eta}_1^{[m]}, \hat{\eta}_2^{[m-1]}, \hat{\eta}_3^{[m-1]}, \ldots, \hat{\eta}_{K-1}^{[m-1]}, \hat{\eta}_K^{[m-1]}\right) \xrightarrow{\text{calculate}} \mathbf{u}_2^{[m-1]} \xrightarrow{\text{fit and update}} \hat{f}_2^{[m]} \\
&\qquad \frac{\partial}{\eta_3}\rho\left(y, \hat{\eta}_1^{[m]}, \hat{\eta}_2^{[m]}, \hat{\eta}_3^{[m-1]}, \ldots, \hat{\eta}_{K-1}^{[m-1]}, \hat{\eta}_K^{[m-1]}\right) \xrightarrow{\text{calculate}} \mathbf{u}_3^{[m-1]} \xrightarrow{\text{fit and update}} \hat{f}_3^{[m]} \\
&\qquad\qquad \ldots \\
&\quad \frac{\partial}{\eta_{K-1}}\rho\left(y, \hat{\eta}_1^{[m]}, \hat{\eta}_2^{[m]}, \hat{\eta}_3^{[m]}, \ldots, \hat{\eta}_{K-1}^{[m-1]}, \hat{\eta}_K^{[m-1]}\right) \xrightarrow{\text{calculate}} \mathbf{u}_{K-1}^{[m-1]} \xrightarrow{\text{fit and update}} \hat{f}_{K-1}^{[m]} \\
&\qquad \frac{\partial}{\eta_K}\rho\left(y, \hat{\eta}_1^{[m]}, \hat{\eta}_2^{[m]}, \hat{\eta}_3^{[m]}, \ldots, \hat{\eta}_{K-1}^{[m]}, \hat{\eta}_K^{[m-1]}\right) \xrightarrow{\text{calculate}} \mathbf{u}_K^{[m-1]} \xrightarrow{\text{fit and update}} \hat{f}_K^{[m]}
\end{aligned}$$

Note that this algorithm resembles the backfitting strategy by 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 $\mathbf{m}_{\text{stop}} = m_{\text{stop},1}, \ldots, m_{\text{stop},K}$. 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 $\mathbf{m}_{\text{stop}}$ 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 of the parameters. To do this, we do what is often called a grid search, i.e., 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. For a schematic overview of this cyclical multidimensional boosting algorithm, see Algorithm 5.

grid-search

3.11.1 Grid search cross-validation in gradient boosting

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

algo:multi-cyclical

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,

$$\hat{\boldsymbol{\eta}} = \underset{\boldsymbol{\eta}}{\operatorname{argmin}} R(\boldsymbol{\eta}) = \underset{\boldsymbol{\eta}}{\operatorname{argmin}} \sum_{i=1}^{n} \rho(y_i, \boldsymbol{\eta})\big|_{\boldsymbol{\eta}=\boldsymbol{\eta}(x_i)}. \tag{3.71}$$

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.

initialization

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

4. Increase m by 1.

5. Set k to 0.

cyclic-first

6. Increase k by 1. If $m > m_{\text{stop},k}$, go to step **??**. 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

$$\mathbf{u}_k^{[m-1]} = \left(-\frac{\partial}{\partial \eta_k} \rho(y_i, \boldsymbol{\eta}_i)\big|_{\boldsymbol{\eta}=\hat{\boldsymbol{\eta}}_i^{[m-1]}} \right)_{i=1}^{N} \tag{3.72}$$

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

$$\hat{h}_{k,1}^{[m]}, \hat{h}_{k,2}^{[m]}, \ldots, \hat{h}_{k,p}^{[m]} \tag{3.73}$$

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

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

$$j_k^{[m]} = \underset{j \in \{1,2,\ldots,p\}}{\operatorname{argmin}} \sum_{i=1}^{N} \left(u_{k,i} - \hat{h}_{k,j}^{[m]} \right)^2. \tag{3.74}$$

cyclic-last

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

$$\hat{f}_{k,j_k^{[m]}}^{[m]} \leftarrow \hat{f}_{k,j_k^{[m]}}^{[m-1]} + \nu \cdot \hat{h}_{j_k^{[m]}}^{[m]}, \tag{3.75}$$

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,

$$\hat{f}_{k,j}^{[m]} \leftarrow \hat{f}_{k,j}^{[m-1]}. \tag{3.76}$$

Then update the full model thus far.

10. Repeat steps 6 to 9 until $m = \max(m_{\text{stop},1}, \ldots, m_{\text{stop},K})$.

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

finite grid with N_k points, such that we obtain

$$m_{\min,k} = m_{1,k} < m_{2,k} < \ldots < m_{N_k-1,k} < m_{N_k,k} = m_{\max,k}, \tag{3.77}$$

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 $m_{\min,k} = 1$ and $m_{\max,k} = 10$ 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 $\mathbf{m}$, enumerated below:

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

We want to find the best configuration $\mathbf{m}$, i.e., we want to find the optimum of the hyperplane $CV(\mathbf{m})$. Like in subsection 3.9.2, we must calculate the estimate of the cross-validated prediction error for each given configuration $\mathbf{m}$, obtaining the prediction error $CV(\mathbf{m})$. We choose $\mathbf{m}_{\text{stop}}$ to be the minimizer of this error,

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

Using boosting, we may obtain estimates of $\text{CV}(\mathbf{m})$ for all $\text{CV}(\mathbf{m})$ 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 $\mathbf{m} = (m_{i_1,1}, m_{i_2,2}, m_{i_3,3})$, where $m_{\min,k} \leq i_k \leq m_{\max,k}$ 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 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 $\mathbf{m}$, i.e.,

$$\begin{aligned}
&(m_{1,1}, m_{1,2}) \\
&(m_{1,1}, m_{2,2}) \\
&\dots \\
&(m_{1,1}, m_{10,2}) \\
&(m_{2,1}, m_{1,2}) \\
&\dots \\
&(m_{2,1}, m_{10,2}) \\
&\dots \\
&(m_{10,2}, m_{10,2}) \,,
\end{aligned}$$

and do a boosting run for each such. Like in subsection 3.9.2, we choose

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

where

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

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

3.12 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 3.11.1, 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).

3.12.1 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*.

Add empirical proof?

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,

$$\hat{h}_{k,j}(\cdot), \tag{3.78}$$

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

$$\hat{\boldsymbol{\eta}}_k^{[m]} = \hat{\boldsymbol{\eta}}_k^{[m-1]} + \nu \cdot \hat{h}_{k,j}^{[m]}(\cdot). \tag{3.79}$$

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

$$R\left(\hat{\boldsymbol{\eta}}_k^{[m]}\right). \tag{3.80}$$

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

$$\Delta\rho_k = R\left(\hat{\boldsymbol{\eta}}^{[m-1]}\right) - R\left(\hat{\boldsymbol{\eta}}_k^{[m]}\right), \tag{3.81}$$

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.,

$$k^{[m]} = \underset{k \in \{1,2,\ldots,K\}}{\operatorname{argmin}} \Delta\rho_k. \tag{3.82}$$

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

$$\hat{\boldsymbol{\eta}}^{[m]} \leftarrow \hat{\boldsymbol{\eta}}_k^{[m]} = \hat{\boldsymbol{\eta}}^{[m-1]} + \nu \cdot \hat{h}_{k,j_k^{[m]}}^{[m]}(\cdot), \tag{3.83}$$

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

$$\hat{\eta}_k^{[m]} \leftarrow \hat{\eta}_k^{[m-1]}. \tag{3.84}$$

3.12.2 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 which minimizes the RSS, i.e.,

$$j^{[m]} = \underset{j}{\operatorname{argmin}} \sum_{i=1}^{N} (u_{k,i} - (f^{[m]}(x_i) + \hat{h}(\mathbf{x}_i)))^2. \tag{3.85}$$

This is, however, not the same criterion that is used to choose between parameters. 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 that case, we instead choose the component-wise learner for each parameter which minimizes the outer loss function, i.e.,

$$j^{[m]} = \underset{j}{\operatorname{argmin}} \sum_{i=1}^{N} \rho(y_i, f(x_i)). \tag{3.86}$$

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. This is unlike the cyclical algorithm, which needs a multidimensional grid search.

algo:multi-noncyclical

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,
$$\hat{f} = \underset{f}{\operatorname{argmin}} R(f). \tag{3.87}$$
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 $-\frac{\partial \rho}{\partial \hat{f}_k}$ and evaluate at $\hat{f}^{(m-1)}(x_i), i = 1, \ldots, N$, yielding negative gradient vector
$$\mathbf{u}_k^{(m-1)} = \left(-\frac{\partial}{\partial \hat{f}_k} \rho(y_i, \hat{f}^{(m-1)}(x_i)) \right)_{i=1}^N \tag{3.88}$$
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 $\mathbf{u}_k^{(m-1)}$.
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
   $$j^* = \underset{j \in 1, \ldots, J_k}{\operatorname{argmin}} \sum_{i=1}^N (u_k^{(i)} - \hat{h}_{kj}(x^{(i)}))^2 \tag{3.89}$$
   - the outer loss, i.e., the loss function after the potential update,
   $$j^* = \underset{j \in 1, \ldots, J_k}{\operatorname{argmin}} \sum_{i=1}^N \rho \left(y^{(i)}, \hat{f}^{(m-1)}(x^{(i)}) + \nu \cdot \hat{h}_{kj}(x^{(i)}) \right) \tag{3.90}$$
10. Compute the possible improvement of this update regarding the outer loss,
$$\Delta\rho_k = \sum_{i=1}^N \rho \left(y^{(i)}, \hat{f}^{(m-1)}(x^{(i)}) + \nu \cdot \hat{h}_{kj^*}(x^{(i)}) \right) \tag{3.91}$$
11. Update, depending on the value of the loss reduction, $k^* = \operatorname{argmin}_{k \in 1, \ldots, K} \Delta\rho_k$
$$\hat{f}_{k^*}^{(m)} = \hat{f}_{k^*}^{(m-1)} + \nu \cdot \hat{h}_{k^* j^*}(x), \tag{3.92}$$
while for all $k \neq k^*$,
$$\hat{f}_{k^*}^{(m)} = \hat{f}_{k^*}^{(m-1)}. \tag{3.93}$$
12. Repeat steps 4 to 11 until $m = m_{\text{stop}}$.
13. Return $\hat{f}(\cdot) = \hat{f}_M(\cdot) = \sum_{m=0}^M f_m(\cdot)$.

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 GAMLSSBoost Algorithm

Response has a distribution

$$f_{\text{dens}}(y_i|\boldsymbol{\theta}_i), \tag{4.1}$$

depending on K distribution parameters

$$\boldsymbol{\theta}_i = (\theta_{i,1}, \theta_{i,2}, \ldots, \theta_{i,K})^T . \tag{4.2}$$

Each distribution parameter $\theta_k, k = 1, 2, \ldots, K$ has its own known monotonic link function

$$g_k(\cdot). \tag{4.3}$$

A GAMLSS is given by the equations

$$g_k(\theta_k) = \beta_{0,\theta_k} + \sum_{j=1}^{p_k} f_{j,\theta_k}(x_{k,j}) = \eta_{\theta_k}, \tag{4.4}$$

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

$$f_{j,\theta_k}(x_{k,j}) = x_{k,j}\beta_{k,j}. \tag{4.5}$$

Finally, η_{θ_k} is the additive predictor for θ_k. We see that the GAMLSS reduces to a GAM (Hastie and Tibshirani, 1990) in the case where the distribution parameter vector is simply

$$\boldsymbol{\theta}_i = \mu, \tag{4.6}$$

i.e., the mean.

4.2 FHTBoost

The first-hitting-time model with Wiener processes, as shown, leads to inverse Gaussian lifetimes. As in the usual regression scheme (see subsection 2.3.9) for this setup, we we have $K = 2$ distribution parameters,

$$\boldsymbol{\theta}_i = (\theta_1, \theta_2)^T = (y_0, \mu)^T. \tag{4.7}$$

We choose link functions

$$g_1(x) = \log(x) \tag{4.8}$$

and

$$g_2(x) = \text{identity}(x) = x, \tag{4.9}$$

for parameters y_0 and μ, respectively.

We apply the GAMLSSBoost algorithm shown previously (see ...) to this setup. The loss function of interest is quite naturally the negative log likelihood of the censored inverse Gaussian distribution, derived in ..., i.e.,

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

(See Appendix for the full expression!) Given an estimated additive predictor $\hat{\boldsymbol{\eta}}$, we get the corresponding estimated distribution parameters by transforming the additive predictors via the inverse of their link functions. Thus,

$$y_0 = \theta_1 = g_1^{-1}(\eta_1) = \exp(\eta_1), \tag{4.10}$$

and

$$\mu = \theta_2 = g_2^{-2}(\eta_2) = \eta_2. \tag{4.11}$$

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 of course need to have the negative gradient of the loss function, i.e., the negative derivative. The negative partial derivative of the loss function is equal to the (positive) derivative of the (positive) loss function, with regard to each parameter. These are

$$p_1(\eta_1, \eta_2) \coloneqq -\frac{\partial}{\partial \eta_1}\rho(y_i, \boldsymbol{\theta}) = \frac{\partial}{\partial \eta_1} l(\exp(\eta_1), \eta_2) \tag{4.12}$$

and

$$p_2(\eta_1, \eta_2) \coloneqq -\frac{\partial}{\partial \eta_2}\rho(y_i, \boldsymbol{\theta}) = \frac{\partial}{\partial \eta_1} l(\exp(\eta_1), \eta_2). \tag{4.13}$$

(See Appendix for the full expressions for these!)

Our loss function is the negative of the log-likelihood in (A.5), so

$$\begin{aligned}\rho(y, \hat{\boldsymbol{\eta}}) &= -l(\hat{y}_0, \hat{\mu}) \\ &= -\delta_i \left(\ln \hat{y}_0 - \frac{1}{2}\ln\left(2\pi\sigma^2 t_i^3\right) - \frac{\left(\hat{y}_0 + \hat{\mu} t_i\right)^2}{2\sigma^2 t_i} \right) \\ &\quad - (1 - \delta_i) \ln\left(\Phi\left(\frac{\hat{\mu} t_i + \hat{y}_0}{\sqrt{\sigma^2 t_i}} \right) - \exp\left(-\frac{2\hat{y}_0\hat{\mu}}{\sigma^2} \right) \Phi\left(\frac{\hat{\mu} t_i - \hat{y}_0}{\sqrt{\sigma^2 t_i}} \right) \right)\end{aligned} \tag{4.14}$$

{eq:fht-loglik}

4.3 Simulation of survival data

We wish to simulate survival times $t_i, i = 1, \ldots, N$ with censoring. We first draw survival times $\tilde{t}_i$ from some survival time distribution $f(\cdot)$. If this distribution has a closed form probability distribution function, we can draw from it directly. If not, we might use some an inverse sampling method, e.g. by drawing unit exponentials and using a corresponding transformation.

To censor the data, we draw censoring times $W_i \sim f(\cdot), i = 1, \ldots, N$, from a more right-tailed distribution, meaning we want to get many, but not all, W_i's to be larger than the $\tilde{t}_i$'s. We let the observed survival times then be $t_i = \min(\tilde{t}_i, W_i)$. The corresponding observed indicator, δ_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 $(t_i, \delta_i), i = 1, \ldots, N$. Note that this scheme incorporates independent censoring: The censoring time is independent of the survival times.

algo:FHT-sim

Algorithm 7 Generating survival data from Inverse Gaussian FHT distribution

1. Given design matrices $\mathbf{X}$, $\mathbf{Z}$ and true parameter vectors $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$.
2. Link covariates and parameters using link functions
$$\ln y_0 = \boldsymbol{\beta}^T \mathbf{X}$$
$$\mu = \boldsymbol{\gamma}^T \mathbf{Z}.$$
3. Draw N survival times $(t_i)_{i=1}^N$ from $\mathrm{IG}(\mu, y_0)$.
4. Draw a censoring time W from some distribution which is independent of the data.
5. Right censor data by choosing $\widetilde{t}_i = \min(t_i, W)$. The indicator on whether observation i was observed or not is then $\delta_i = I(\widetilde{t}_i = t_i)$.
6. The simulated data set is $(\widetilde{t}_i, \delta_i)_{i=1,\ldots,N}$.

4.4 Algorithm

We apply the component-wise boosting algorithm 8 with loss function $\rho(\mu, \mathbf{y}0) = -\log \mathrm{L}\, y_0, \mu$. We differentiate the loss function with respect to these two and get .... For more details on the derivation, see A.

Maybe use b instead of y_0, to not get subscript chaos?

We might call this cyclical boosting.

4.4.1 Boost in same

Another way to do this is to only boost one component in each iteration. The component might be corresponding to X, or it might be corresponding to Z.

4.4.2 Derivatives not on same scale

Pass.

algo:fhtboost

Algorithm 8 FHT Boost with twodimensional loss function

1. Initialize the n-dimensional vectors $\hat{y}_0^{[0]}, \hat{\mu}^{[0]}$ with the maximum likelihood estimates as offset values, i.e., $\hat{y}_0^{[0]}, \hat{\mu}^{[0]} = \text{argmin}_{y_0,\mu} \rho(\cdot, \cdot)$.

2. For both components of the loss function, we specify linear base learners. In particular, a component-wise base learner which can be used for each of the p variables used in $\mathbf{X}$ corresponding to y_0 and the d variables in $\mathbf{Z}$ corresponding to μ. Like earlier, the base learner takes one input variable and has one output variable.

3. Set $m = 0$ and $\nu = 0.1$.

4. Increase m by 1.

   a) If $m > m_{\text{stop},y_0}$, proceed to step 4 e). If not, compute the negative partial derivative $-\frac{\partial \rho}{\partial y_0}$ and evaluate at $\hat{f}^{[m-1]}(X_i, Z_i) = \left(\hat{y}_0^{[m-1]}(X_i), \hat{\mu}^{[m-1]}(Z_i)\right)_{i=1,\ldots,n}$. This yields the negative gradient vector $U_{y_0}^{[m-1]} = \left(U_{i,y_0}^{[m-1]}\right)_{i=1,\ldots,n} := \left(-\frac{\partial}{\partial y_0}\rho\left(Y_i, \hat{f}^{[m-1]}(X_i, Z_i)\right)\right)_{i=1,\ldots,n}$.

   b) Fit the negative gradient vector $U_{y_0}^{[m-1]}$ to each of the p components of $\mathbf{X}$ separately (i.e. to each predictor variable) using the base learners specified in step 2. This yields p vectors of predicted values, where each vector is an estimate of the negative gradient vector $U_{y_0}^{[m-1]}$.

   c) Select the component of $\mathbf{X}$ which best fits $U_{y_0}[m-1]$ according to R^2. Set $\hat{U}_{y_0}^{[m-1]}$ equal to the fitted values of the corresponding best model fitted in the previous step.

   d) Update $\hat{y}_0^{[m-1]} \leftarrow \hat{y}_0^{[m-1]} + \nu \hat{U}_{y_0}^{[m-1]}$.

   e) If $m > m_{\text{stop},\mu}$, proceed to step 4 j). If not, compute the negative partial derivative $-\frac{\partial \rho}{\partial \mu}$ and evaluate at $\hat{f}^{[m-1]}(X_i, Z_i) = \left(\hat{y}_0^{[m-1]}(X_i), \hat{\mu}^{[m-1]}(Z_i)\right)_{i=1,\ldots,n}$. This yields the negative gradient vector $U_{\mu}^{[m-1]} = \left(U_{i,\mu}^{[m-1]}\right)_{i=1,\ldots,n} := \left(-\frac{\partial}{\partial \mu}\rho\left(Y_i, \hat{f}^{[m-1]}(X_i, Z_i)\right)\right)_{i=1,\ldots,n}$.

   f) Fit the negative gradient vector $U_{\mu}^{[m-1]}$ to each of the p components of $\mathbf{Z}$ separately (i.e. to each predictor variable) using the base learners specified in step 2. This yields d vectors of predicted values, where each vector is an estimate of the negative gradient vector $U_{\mu}^{[m-1]}$.

   g) Select the component of $\mathbf{Z}$ which best fits $U_{\mu}[m-1]$ according to R^2. Set $\hat{U}_{\mu}^{[m-1]}$ equal to the fitted values of the corresponding best model fitted in the previous step.

   h) Update $\hat{\mu}^{[m-1]} \leftarrow \hat{\mu}^{[m-1]} + \nu \hat{U}_{\mu}^{[m-1]}$.

   i) Update $\hat{f}^{[m]} \leftarrow \hat{f}^{[m-1]}$.

   j) If $m > \max(m_{\text{stop},y_0}, m_{\text{stop},\mu})$, go to step 5. If not, repeat step 4.

5. Return $\hat{f}^{[m]}$.

4.4.3 Changing the intercept in each iteration

Another way to do this is to only boost one component in each iteration. The component might be corresponding to X, or it might be corresponding to Z.

4.5 Simulation experiments

In this section, I will discuss how I tried validating the boosting method I have developed. While working with implementing the algorithm, to see if it worked, I first used an example with low dimensions. In low dimensions, it's feasible to find the joint maximum likelihood numerically. After confirming the method works as it should, we can go to more complicated examples.

4.6 Simulation setup

We do simulations where we draw observations from the Inverse Gaussian distribution, i.e., we simulate lifetimes from the first hitting time model with Wiener process as the health process. We use algorithm (4.6) to do this. We will have two scenarios: One with no correlation, and one with a lot of correlation. To simulate the covariate matrices X and Z we will use algorithm (9), which is a method for simulating clinical and gene data together. We imagine X, corresponding to $\boldsymbol{\beta}$, be gene expressions, whereas Z, corresponding to $\boldsymbol{\gamma}$ 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.

algo:clinical-sim

One run consists of first drawing data (with a specific seed to ensure reproducibility), i.e., we draw covariate matrices $\mathbf{X}$, $\mathbf{Z}$ from . $\mathbf{X}$ is of size $N \times p$ and $\mathbf{Z}$ 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 $\mathbf{y}_0$ and μ of initial value of the health process, and drift, respectively. Then we draw from the Inverse Gaussian distribution according to , obtaining N right-censored lifetimes, i.e., N tuples $(\widetilde{t}_i, \delta_i)_{i=1,\ldots,N}$. 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 training set of size N_{test}. The data here is drawn in the exact same manner as the training data, here also with a specific seed.

algo:FHT-sim

Algorithm 9 Generating clinical and gene expression data

algo:clinical-sim

1. Lorem ipsum.

4.6.1 Small example

Let parameter vectors be two dense $\boldsymbol{\beta} = (2, 0.1, 0.2)$ and $\boldsymbol{\gamma} = (-1, -0.1, 0.1)$. Let X and Z be such and such, drawn from a beta distribution. We simulate

Figure 4.1: Kaplan-Meier plot of small example

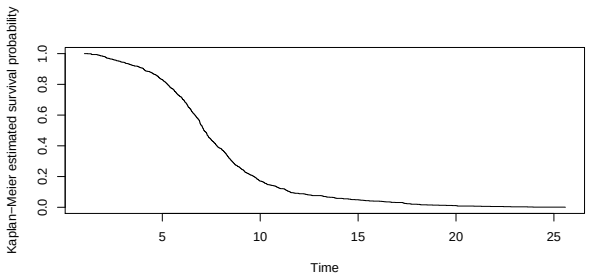
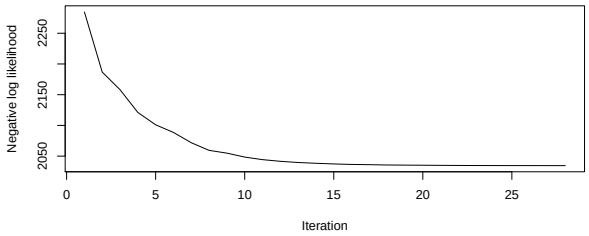


Figure 4.2: Log-likelihood



data using Algorithm 4.6, with the censoring time W being drawn from a distribution exp(0.1). The resulting survival times have the following Kaplan-Meier plot. We use cross validation to find a suitable iteration number m_{stop}, and find it to be 28. We then run our algorithm with that number of iterations. Below is a plot of the negative log likelihood of the data (in-sample loss) as a function of iteration number. The final $\hat{\boldsymbol{\beta}}$ is $(1.968, 0.103, 0.180)$, and the final $\hat{\boldsymbol{\gamma}}$ is $(-0.964, -0.082, 0.062)$. The parameters found by numerically maximizing the joint maximum likelihood are also included. Summarized in the table below.

parameter	true	estimated
β_0	2.0	1.968
β_1	0.1	0.103
β_2	0.2	0.180
γ_0	-1.0	-0.964
γ_1	-0.1	-0.082
γ_2	0.1	0.062

As we can see, our boosting method recovers the original parameters quite well. This is of course with data coming from the exact same kind of model.

4.7 Brier scores on survival data

In order to assess predictive performance, we need what's called a proper scoring rule: One which is 1 if the actual data is inserted as predictions.

4.7.1 Brier scores and R^2 measure

The Brier score (Brier, 1950) was first introduced as a way to measure the accuracy of weather forecasts. While it is also able to handle multiple categories, we will here give the definition for the binary case. We start by assuming that we are in a situation with no censoring, and that we have m individuals in a test set. We denote their observed survival times by t_i, and their covariate vector as $\mathbf{x}_i$, as usual, with $i = 1, \ldots, m$. The Brier score aims at evaluating how well the estimated patient specific survival probability $\hat{\pi}(t^*|\mathbf{x})$, obtained from a prediction model, is able to predict the event status $I(t > t^*)$ of an individual at a given time t^*. The error made in predicting the event status $I(t > t^*)$ for a patient in the test set can be given as

$$\begin{aligned} BS(t^*) &= \frac{1}{M}\sum_{i=1}^{m}\left(I(t_i > t^*) - \hat{\pi}(t^*|\mathbf{x}_i)\right)^2 \\ &= \frac{1}{M}\sum_{i=1}^{m}\left[\hat{\pi}(t^*|\mathbf{x}_i)^2 I(t_i \leq t^*) + (1 - \hat{\pi}(t^*|\mathbf{x}_i))(I(t_i > t^*)\right]. \end{aligned}$$

In the first formulation, the Brier score looks like a version of an RSS measure, where one sums the squared error between the observed event and the estimated probability. In the case of censored data, the above is of course not enough. The Brier score was adapted to handle censored survival times by Graf et al. (1999), assuming independent censoring. They showed that the loss of information due to censoring can be accounted for by using an inverse probability of censoring weighting (Bøvelstad and Borgan, 2011). This Brier score for censored data is defined as

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

where $\hat{G}$ is the Kaplan-Meier estimate of the censoring distribution, defined as

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

where $Y_i(t)$ is an indicator of whether individual i is at risk at time t. This score may also be used to define an R^2 measure, where one can benchmark the performance of a fitted model to a so-called null model, i.e., one where each regression coefficient is set to zero. A measure of explained variation can be found by calculating the gain in accuracy when adding covariates. Thus we define the Brier R^2 measure as

$$R^2_{\text{Brier}}(t^*) = 1 - \frac{BS^c(t^*)}{BS^c_0(t^*)},$$

where $BS^c_0(t^*)$ is the Brier score for the null model, in other words, one which assigns equal probability to all individuals. One advantage with R^2_{Brier} is that it adjusts for variation due to the specific data under study, which the Brier score itself does not (Bøvelstad and Borgan, 2011).

Chapter 5

Simulations

To see if the FHTBoost algorithm developed actually works, we need to test it on simulated data. In this chapter we describe two scenarios, and for each, use the algorithm to estimate a model, and see how it performs. The scenarios are both thought to be realistic scenarios where one has clinical measurements, in a high-dimensional covariate matrix $\mathbf{X}$, as well as gene expressions, in a low-dimensional covariate matrix $\mathbf{Z}$. We link each covariate to a parameter in a parameter vector, where $\mathbf{X}$ corresponds to $\boldsymbol{\beta}$, and $\mathbf{Z}$ corresponds to $\boldsymbol{\gamma}$. In each parameter vector, we set a small number of parameters to a non-zero value, and all the rest to zero. This means that only a very small number of covariates have an effect. Given parameter vectors and covariate matrices, we can calculate a specific y_0 and a μ for each individual. We draw an observation from the inverse gaussian distribution with these parameters.

We use the FHTBoost algorithm to estimate the parameter vector

We have discussed the importance of a test set, and of finding an appropriate m_{stop}. Since we are simulating, we can generate a test set by drawing from a seed, and we therefore made the test set quite big, with $N_{\text{test}} = 1000$ observations.

We generate $B \approx 500$ data sets by drawing FHT data according to algorithm X. 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 perform repeated 5-fold cross validations, with 5 repeats, on the training data set. As shown in section ..., this should provide a reasonably stable m_{stop} (near the "true" m_{stop}) for that specific data set. We then estimate a model on the test set, by running FHTBoost with m_{stop} number of iterations. We record metrics on these, concerning variable selection and log-likelihood (loss function) performance.

5.1 Variable selection metrics

As mentioned in section ..., a component-wise algorithm, such as FHTBoost, performs data-driven variable selection. We wish to see how the boosting algorithm performs, does it select informative variables and make sure not to select non-informative ones? In a given estimated boosting model, the model has selected a certain amount of variables. We denote a selected variable as a "positive," or P for short, and a variable which 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 so 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 variables which have been selected but which do not actually have an effect, are false positives, FP. Finally, false negatives, FN, are variables which do actually have an effect, but which were not selected in the boosting model.

5.1.1 Sensitivity

Ideally, this is 1.

$$\text{Sensitivity} = \frac{TP}{P} \tag{5.1}$$ {eq:sensitivity}

5.1.2 Specificity

Ideally, this is 1.

$$\text{Specificity} = \frac{TN}{N} \tag{5.2}$$ {eq:specificity}

5.1.3 Accuracy

Ideally, this is 1.

$$\text{ACC} = \frac{TP + TN}{P + N} \tag{5.3}$$ {eq:sensitivity}

5.1.4 False positive rate

Ideally, this is 0.

$$\text{FPR} = \frac{FP}{N}. \tag{5.4}$$ {eq:sensitivity}

5.1.5 False negative rate

Ideally, this is 0.

$$\text{FNR} = \frac{FN}{P} \tag{5.5}$$ {eq:sensitivity}

5.2 Large simulation with uncorrelated matrices

Here, N is 500. We let $\boldsymbol{\beta}$ be a large vector of size $p = 10001$, and $\boldsymbol{\gamma}$ be a small vector of size $d = 16$. Specifically, we set the intercept term in $\boldsymbol{\beta}$ to be 2.0, and the first 35 elements to be 0.1. We set the rest to be 0. For $\boldsymbol{\gamma}$, we set the intercept term to be -1, and in similar fashion, let the first 5 elements have a

Table 5.1: Summary of results

table:non-correlated-with-intercept-summary

Measure	Mean	Standard deviation	Minimum	Maximum
Deviance	91.955	41.760	-7.191	233.563
m_{stop}	15.823	6.435	2	39
Log-likelihood	-2325.746	20.363	-2382.825	-2270.023
Null model log-likelihood	-2371.723	3.935	-2396.848	-2368.769

non-zero value of -0.1. Here also we set the remaining 10 elements to be 0. So,

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

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

We draw X and Z from

To make a test set, we draw $N_{\text{test}} = 1000$ lifetimes from a distribution with the same parameters, but of course with a unique seed.

add Kaplan-Meier plot

With this exact setup, we run a simulation experiment $B = 500$ times, where we at the beginning of each simulation set the seed, via `set.seed(seed)` to be $b = 1, \ldots, B$. We first generate matrices X and Z, and simulate FHT times from the algorithm above. We then run cross validation on this data set to find the optimal iteration number m_{stop}. Below is a box plot of the resulting times. We then run a boosting algorithm with m_{stop} steps on the training set, and use the resulting model on the test set.

5.2.1 Deviance

The difference in deviance between a fitted model and the null model containing no covariates is given by

$$d = -2\left(l^{\text{test}}(\beta_{\text{train}}) - l^{\text{test}}(0)\right),$$

where $l^{\text{test}}(\beta)$ is the likelihood attained with covariate vector β on the test set.

5.2.2 Boosting with changing the intercept

See Table 5.1, Table 5.2, and 5.3.

5.2.3 Boosting *without* changing the intercept

See Table 5.4, Table 5.5, and 5.6.

5.3 Large simulation with correlated matrices

5.3.1 Setup

Lorem ipsum.

Table 5.2: Result for y_0, β

table:non-correlated-with-intercept-y0

Measure	Mean	Standard deviation
Sensitivity	0.187	0.092
Specificity	1.000	0.000
Accuracy	0.997	0.000
False positive rate	0.000	0.000
False negative rate	0.813	0.092

Table 5.3: Result for μ, γ

table:non-correlated-with-intercept-mu

Measure	Mean	Standard deviation
Sensitivity	0.729	0.249
Specificity	0.944	0.109
Accuracy	0.872	0.091
False positive rate	0.056	0.109
False negative rate	0.271	0.249

Table 5.4: Summary of results

table:non-correlated-no-intercept-summary

Measure	Mean	Standard deviation	Minimum	Maximum
Deviance	130.109	40.699	5.710	255.184
m_{stop}	63.791	26.526	2.000	160.000
Log-likelihood	-2306.669	21.512	-2370.614	-2241.724
Null model log-likelihood	-2371.723	3.935	-2396.848	-2368.769

Table 5.5: Result for y_0, β

table:non-correlated-no-intercept-y0

Measure	Mean	Standard deviation
Sensitivity	0.453	0.162
Specificity	0.997	0.002
Accuracy	0.995	0.001
False positive rate	0.003	0.002
False negative rate	0.547	0.162

Table 5.6: Result for μ, γ

table:non-correlated-no-intercept-mu

Measure	Mean	Standard deviation
Sensitivity	0.953	0.119
Specificity	0.640	0.291
Accuracy	0.745	0.186
False positive rate	0.360	0.291
False negative rate	0.047	0.119

Table 5.7: Summary of results

table: correlated-intercept-summary

Measure	Mean	Standard deviation	Minimum	Maximum
Deviance	57.769	47.218	-87.558	203.403
m_{stop}	20.041	12.081	2	65
Log-likelihood	-2230.488	25.796	-2342.571	-2161.701
Null model log-likelihood	-2259.373	13.447	-2350.838	-2250.039

Table 5.8: Result for y_0, β

table: correlated-intercept-y0

Measure	Mean	Standard deviation
Sensitivity	0.156	0.084
Specificity	0.999	0.000
Accuracy	0.996	0.000
False positive rate	0.001	0.000
False negative rate	0.844	0.084

Table 5.9: Result for μ, γ

table: correlated-intercept-mu

Measure	Mean	Standard deviation
Sensitivity	0.235	0.198
Specificity	0.854	0.141
Accuracy	0.607	0.059
False positive rate	0.146	0.141
False negative rate	0.765	0.198

5.3.2 Boosting with changing the intercept

See table 5.7, table 5.8, and 5.9.

5.3.3 Boosting *without* changing the intercept

See table 5.10, table 5.11, and 5.12.

Table 5.10: Summary of results

table: correlated-no-intercept-summary

Measure	Mean	Standard deviation	Minimum	Maximum
Deviance	58.785	46.128	-73.525	223.132
m_{stop}	51.1	24.4	2	148
Log-likelihood	-2229.980	30.386	-2351.289	-2159.627
Null model log-likelihood	-2259.373	13.447	-2350.838	-2250.039

table: correlated-no-intercept-y0

Table 5.11: Result for y_0, β

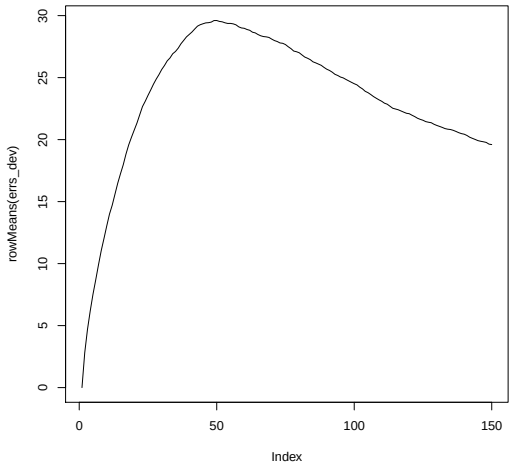
Measure	Mean	Standard deviation
Sensitivity	0.204	0.082
Specificity	0.998	0.001
Accuracy	0.995	0.001
False positive rate	0.002	0.001
False negative rate	0.796	0.082

table: correlated-no-intercept-mu

Table 5.12: Result for μ, γ

Measure	Mean	Standard deviation
Sensitivity	0.606	0.264
Specificity	0.551	0.245
Accuracy	0.573	0.086
False positive rate	0.449	0.245
False negative rate	0.394	0.264

Figure 5.1: YYY



Appendices

Appendix A

Appendix 1: Differentiating the IG FHT

appendix

First we have the likelihood,

$$L(y_0,\mu)=\prod_{i=1}^{n}\left(\frac{y_0}{\sqrt{2\pi\sigma^2 t_i^3}}\exp\left[-\frac{(y_0+\mu t_i)^2}{2\sigma^2 t_i}\right]\right)^{\delta_i} \times\left[1-\Phi\left(-\frac{y_0+\mu t_i}{\sqrt{\sigma^2 t_i}}\right)-\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t_i-y_0}{\sqrt{\sigma^2 t_i}}\right)\right]^{1-\delta_i}, \tag{A.1}$$

{eq:fht-lik}

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

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

and so in particular,

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

and thus we can rewrite (A.1) as

$$L(y_0,\mu)=\prod_{i=1}^{n}\left(\frac{y_0}{\sqrt{2\pi\sigma^2 t_i^3}}\exp\left[-\frac{(y_0+\mu t_i)^2}{2\sigma^2 t_i}\right]\right)^{\delta_i} \times\left[\Phi\left(\frac{y_0+\mu t_i}{\sqrt{\sigma^2 t_i}}\right)-\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t_i-y_0}{\sqrt{\sigma^2 t_i}}\right)\right]^{1-\delta_i}. \tag{A.4}$$

{eq:fht-lik-proper}

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

$$l(y_0,\mu)=\sum_{i=1}^{n}\delta_i\left(\ln y_0-\frac{1}{2}\ln\left(2\pi\sigma^2 t_i^3\right)-\frac{(y_0+\mu t_i)^2}{2\sigma^2 t_i}\right) +(1-\delta_i)\ln\left(\Phi\left(\frac{\mu t_i+y_0}{\sqrt{\sigma^2 t_i}}\right)-\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t_i-y_0}{\sqrt{\sigma^2 t_i}}\right)\right) \tag{A.5}$$

{eq:fht-loglik}

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

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

and

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

So we get

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

Thus we see that the partial derivatives are

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

and

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

We take these one by one

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

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

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

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

Hence

$$\begin{aligned}&\frac{\partial}{\partial y_0} l(y_0,\mu) \\ &= \sum_{i=1}^{n}\left(\delta_i\left(\frac{1}{y_0} - \frac{y_0+\mu t_i}{\sigma^2 t_i}\right) + (1-\delta_i)\left[\frac{1}{\sqrt{\sigma^2 t_i}}\phi\left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}}\right) + \frac{2\mu}{\sigma^2}\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right)\right.\right. \\ &\left.\left. + \frac{1}{\sqrt{\sigma^2 t_i}}\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right)\right]\left[\Phi\left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}}\right) - \exp(-\frac{2y_0\mu}{\sigma^2}\Phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right)\right]^{-1}\right)\end{aligned} \tag{A.15}$$

and

$$\frac{\partial}{\partial\mu}l(y_0,\mu)$$
$$=\sum_{i=1}^{n}\left(\delta_i\left(-\frac{y_0+\mu t_i}{\sigma^2}\right)+(1-\delta_i)\left[\frac{t_i}{\sqrt{\sigma^2 t_i}}\phi\left(\frac{\mu t_i+y_0}{\sqrt{\sigma^2 t_i}}\right)+\frac{2y_0}{\sigma^2}\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t_i-y_0}{\sqrt{\sigma^2 t_i}}\right)\right.\right.$$
$$\left.\left.-\frac{t_i}{\sqrt{\sigma^2 t_i}}\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\phi\left(\frac{\mu t_i-y_0}{\sqrt{\sigma^2 t_i}}\right)\right]\left[\Phi\left(\frac{\mu t_i+y_0}{\sqrt{\sigma^2 t_i}}\right)-\exp(-\frac{2y_0\mu}{\sigma^2}\Phi\left(\frac{\mu t_i-y_0}{\sqrt{\sigma^2 t_i}}\right)\right]^{-1}\right) \tag{A.16}$$

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.

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.

chang2010 — Chang, Y.-C. I., Huang, Y., and Huang, Y.-P. (2010). Early stopping in l2boosting. *Computational Statistics & Data Analysis*, 54(10):2203 – 2213.

chhikara1988 — Chhikara, R. (1988). *The Inverse Gaussian Distribution: Theory: Methodology, and Applications*. Statistics: A Series of Textbooks and Monographs. Taylor & Francis.

cox1965 Cox, D. and Miller, H. (1965). *The theory of stochastic processes*. Wiley publications in statistics. Wiley.

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.

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.

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.

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.

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.

gamboost Tutz, G. and Binder, H. (2006). Generalized additive modeling with implicit variable selection by likelihood-based boosting. *Biometrics*, 62(4):961–971.

whitmore1975 Whitmore, G. A. (1975). The inverse gaussian distribution as a model of hospital stay. *Health services research*, 10:297–302.

whitmore1986 Whitmore, G. A. (1986). First-passage-time models for duration data: Regression structures and competing risks. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 35(2):207–219.

whitmore1995 Whitmore, G. A. (1995). Estimating degradation by a wiener diffusion process subject to measurement error. *Lifetime Data Analysis*, 1(3):307–319.

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.