Boosting the First-Hitting-Time Regression Model

Vegard Stikbakke

February 28, 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

Survival analysis

2.1 Survival data and basic definitions

Survival analysis is the field of studying lifetime and time-to-death data. We look at a stochastic variable $T > 0$ which is the time to some event. 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. One example is 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. Typically these trials are for a set amount of time, say, until τ. Luckily, not every patient relapses during that time, and so their time of relapse T_i is not observed. We could throw away these observations, but we at least know that they survived until $t = \tau$. We therefore work with the concept of incomplete data, which we call *censored* data. An observed lifetime $\tilde{T}$ is censored if the actual lifetime T is larger than $\tilde{T}$. We can say that we have a censoring mechanism which works such that the observed $\tilde{T} = \min(T, C)$, where C is a censoring time. In the clinical trial example mentioned, $C = \tau$. We also need a censoring indicator, $D = I(\tilde{T} = T)$, indicating if we have observed the actual event.

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

2.1.1 Censored survival data

If the event has occurred, the indicator d_i is 1.

2.1.2 The survival function $S(t)$

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

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

Here $F(t)$ is the familiar cumulative distribution function. If the derivative $f(t)$ of $F(t)$ exists, the lifetime T has probability distribution function $f(t)$.

2.1.3 The hazard function $\alpha(t)$

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

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

The estimation of the hazard function is hard, and we do not achieve the usual $\sqrt{n}$ convergence.

add citation

Note that

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

and inserting this into the hazard function yields

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

{eq:hfs}

where the probability distribution function $f(t)$ is obtained by its limit definition, and we note that $S'(t)$ is the derivative of $1 - F(t)$, which is $-f(t)$. By integrating the hazard from 0 to time t, we get the cumulative hazard function $A(t) = \int_0^t \alpha(s)\, \mathrm{d}s$,

$$A(t) = -\int_0^t \frac{S'(s)}{S(s)}\, \mathrm{d}s = -\int_0^t \frac{\frac{\mathrm{d}S}{\mathrm{d}f}}{S(s)}\, \mathrm{d}s = -\int_0^t \frac{1}{S(s)}\, \mathrm{d}s = -\log(S(t)). \tag{2.2}$$

{eq:cumulative-hazard}

Given censored survival data $(t_i, d_i), i = 1, \ldots, n$, we introduce the at-risk function $Y(t)$, which is equal to the number of individuals still at risk at time t,

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

where $\#(\cdot)$ is the counting operator over a set. We may then estimate the survival function $S(t)$ by the Kaplan-Meier estimator (Kaplan and Meier, 1958)

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

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

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

2.2 Survival data likelihood regression setup

Given survival data with covariates, $(t_i, d_i, \mathbf{x}_i)$, and parameterized functions $S(t|\mathbf{x}_i, \boldsymbol{\beta})$ and $f(t|\mathbf{x}_i, \boldsymbol{\beta})$ corresponding to a survival distribution, where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)$ is a vector of regression coefficients, we want to set up a likelihood for the data. Assume that the data is independent and identically distributed. We can then use the information about the lifetime distribution, such that the single individual i contributes

$$f(t_i|\mathbf{x}_i) \tag{2.3}$$

{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 rather use the survival distribution. Therefore this observation contributes

$$S(t_i|\mathbf{x}_i) \tag{2.4}$$

{eq:S}

to the likelihood. Of course, since an observation can only be either censored or not censored at the same time, δ_i is either 0 or 1. We can cmombine expressions (2.3) and (2.4) in a way that a single observation contributes

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

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

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} f(t_i|\mathbf{x}_i, \boldsymbol{\beta})^{d_i} S(t_i|\mathbf{x}_i, \boldsymbol{\beta})^{1-d_i}. \tag{2.5}$$

{eq:surv-lik}

Since it is more convenient to work with the log likelihood, we calculate this as well,

$$\begin{aligned} l(\boldsymbol{\beta}) &= \log L(\boldsymbol{\beta}) \\ &= \sum_{i=1}^{n} \left[d_i \log f(t_i|\mathbf{x}_i, \boldsymbol{\beta}) + (1-d_i) \log S(t_i|\mathbf{x}_i, \boldsymbol{\beta}) \right]. \end{aligned} \tag{2.6}$$

{eq:surv-lik}

Note that since $\log S(t) = -A(t)$ and $f(t) = \alpha(t)S(t)$, see (2.2) and (2.1), respectively, (2.6) further simplifies to

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

2.3 Proportional hazards regression

How may we use a covariate vector $\mathbf{x}$ in modelling, say, the hazard rate? A very common model to choose here is that of a proportional hazards model, which assumes

$$\alpha(t|\mathbf{x}) = \alpha_0(t) r(\mathbf{x}|\boldsymbol{\beta}), \tag{2.7}$$

{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 $(t_i, d_i), i = 1, \ldots, n$, we may set up a so-called partial likelihood. For all observations $i = 1, \ldots, n$ with $d_i = 1$, we know that there is an event at time t_i. The probability of the event happening for some individual j is the hazard, i.e., the instantaneous probability of that individual at that time, divided by

the sum of all such hazards for those individuals still alive. Assuming that observations are independent and identically distributed, the partial likelihood for the data is then the product of all such ratios,

$$\text{pl}(\boldsymbol{\beta}) = \prod_{d_i=1} \frac{\Pr(\text{event happens to } i \text{ at time } t_i)}{\sum_{j \in R(t_i)} \Pr(\text{event happens to } j \text{ at time } t_i)} = \prod_{d_i=1} \frac{\alpha_0(t_i) r(\mathbf{x}_i)}{\sum_{j \in R(t_i)} \alpha_0(t_i) r(\mathbf{x}_j)},$$

where we see that the baseline hazard will cancel out, and we are left with just the relative risk functions.

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, 1992). 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 covariates $\mathbf{x}_1$ 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

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

Cox regression is used very much in applied research.

add Cox regression example

2.3.1 Issues with the proportional hazards assumption

Assuming (2.7), i.e. $\alpha(t|\mathbf{x}) = \alpha_0(t) r(x|\boldsymbol{\beta})$, we are making a relatively strong assumption, i.e. that the ratio between the hazard function of two individuals is the same *at all times*. 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, or it is in any case 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

$$\alpha(t|\mathbf{x}) = \beta_0(t) + \beta_1(t) x_1(t) + \beta_2(t) x_2(t) + \ldots + \beta_p(t) x_p(t), \tag{2.8}$$

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.

2.3.2 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.

2.4 First hitting time models or threshold regression

sec:FHT

Blabla.

2.4.1 General idea

So far we have not thought much about how a time-to-event is generated. Instead, we have modelled the hazard rate directly. We have simply said that we have stochastic lifetimes. At one time, an individual is alive, and at a slightly later time, it is perhaps dead. One way to think about how these times are generated is to imagine that each individual has an underlying stochastic process, a health process $Y(t)$, say. Since the process is a function of time, it has a non-negative domain, $t \geq 0$. This health process is not observable, but when it hits a certain boundary set $\mathcal{B}$, the individual dies. $\mathcal{B}$ is also called a barrier or a threshold, depending on what kind of set it is, and what association one wishes to envoke. The lifetime T, then, becomes the time taken by the health process $Y(t)$ to enter the boundary set $\mathcal{B}$. In general, the health process $Y(t)$ takes values in a set $\mathcal{Y}$, with an initial value $y_0 = Y(0)$. The barrier is a subset of this set of values, $\mathcal{B} \in \mathcal{Y}$, with the initial health process value $y_0 \notin \mathcal{B}$. In other words, the lifetime is

$$T = \min_t (t \colon Y(t) \in \mathcal{B}) \tag{2.9}$$

{eq:fht-t}

Models based on this view are called first hitting threshold 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. 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. FHT 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

citation needed

, the duration time of a strike, the failure time of an engineering system

citation needed

citation

, and so on.

Assess lung cancer risk in railroad workers (Lee et al., 2004).

Length of stay as a stochastic process: A general approach and application to hospitalization for schizophrenia (Eaton and Whitmore, 1977).

Model for the duration of a strike (Lancaster, 1972).

Hospital stay (A Whitmore, 1975).

Labour turnover (Whitmore, 1979).

Degradation (Whitmore, 1995).

2.4.2 Choice of the health process

The first hitting time model framework is highly flexible. We have flexibility both in choice of process, boundary and initial value. The most important part is the stochastic process. Examples include Wiener processes, Markov chains, Bernoulli processes, and Gamma processes. We choose to use the

Wiener process, because it has a simple intuition behind it, and it yields a fully parametric regression model.

2.4.3 Wiener process

Let $W(t)$ be a continuous stochastic process 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,

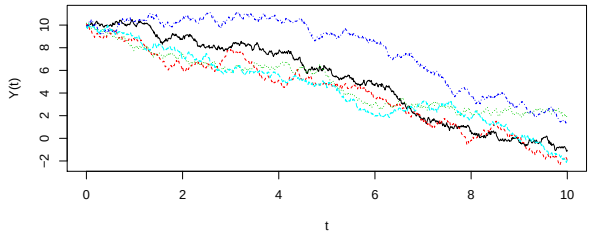
$$Y(t) = y_0 - \mu t + \sigma W(t), \tag{2.10}$$

{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). Figure 2.4.3 shows examples of 5 Wiener process paths with initial value $y_0 = 10$ and negative drift $\mu = 1$.

plot:wiener

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



Clearly, for a Wiener process starting in $y_0 > 0$, with a negative drift, i.e. $\mu > 0$, the movement is markedly in the direction of zero. If σ^2 is small in comparison to the drift, the process will move in almost a straight line, such that $X(t) \approx y_0 - \mu t$. The hitting time will then be nearly a deterministic function of y_0 and μ, $T \approx y_0/\mu$. For a larger relative σ^2, the diffusion part is more dominant and the hitting time thus less predictable (Aalen et al., 2008).

This is a very conceptually appealing model, because it assumes that individuals might have different initial levels, and that also the drift might be different between individuals. It is also attractive because it has closed-form probability and cumulative density functions, and its likelihood is computationally simple. There are no restrictions on the movements of the process, meaning, it is non-monotonic. 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.

2.4.4 FHT with Wiener process and Inverse Gaussian lifetimes

If we choose the stochastic process to be a Wiener process like in (2.10), and we let the boundary be the non-positive numbers, $\mathcal{B} = (-\infty, 0]$, then (2.9) becomes

$$T = \min_t \left(t \colon Y(t) \in \mathcal{B}\right), \tag{2.11}$$

i.e. the the lifetime is the time it takes for the process to first reach a non-positive value. Note that since the Wiener process is continuous, there will not be a difference between $\leq$ and $<$. It can be shown that the first hitting time of the Wiener process follows an inverse Gaussian distribution (Chhikara, 1988), with probability distribution function

$$f(t|y_0, \mu, \sigma^2) = \frac{y_0}{\sqrt{2\pi\sigma^2 t^3}} \exp\left[-\frac{(y_0 + \mu t)^2}{2\sigma^2 t}\right], \tag{2.12}$$

{eq:ig-pdf}

and cumulative distribution function

$$F(t|\mu, \sigma^2, y_0) = \Phi\left[-\frac{\mu t + y_0}{\sqrt{\sigma^2 t}}\right] + \exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left[\frac{\mu t - y_0}{\sqrt{\sigma^2 t}}\right]. \tag{2.13}$$

{eq:ig-cdf}

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 (2.12) 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

$$S(t|\mu, \sigma^2, y_0) = \Phi\left[\frac{\mu t + y_0}{\sqrt{\sigma^2 t}}\right] - \exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left[\frac{\mu t - y_0}{\sqrt{\sigma^2 t}}\right], \tag{2.14}$$

{eq:ig-surv}

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

$$\Phi(x) = \int_{-\infty}^{x} \frac{\exp\left(-y^2/2\right)}{\sqrt{2\pi}}\, \mathrm{d}y, \tag{2.15}$$

and in (2.14) we used the fact that $1 - \Phi(-x) = \Phi(x)$, since the standard normal distribution is symmetric around 0.

2.4.5 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 (2.12) and

the survival function $S(t|\mu, \sigma^2, y_0)$ in (2.14) only depend on these parameters through μ/σ and y_0/σ. Hence, there are only two free parameters. In other words, we can without loss of generality fix one parameter, for instance set σ equal to 1. This is the conventional way to proceed (Lee and Whitmore, 2006). We will use this to make inference.

2.4.6 The shape of the hazard rate

The hazard rate is $\alpha(t) = f(t)/S(t)$ (2.1). 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

2.4.7 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

$$\lim_{t\to\infty} \alpha(t) = \frac{1}{2}\left(\frac{\mu}{\sigma}\right)^2 = 0.5\mu^2 \tag{2.16}$$

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

2.4.8 Regression

citation needed

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 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,

$$\mu(\boldsymbol{\beta}) = \boldsymbol{\beta}^{\mathrm{T}}\mathbf{x} = \sum_{j=1}^{p} \beta_j x_j, \tag{2.17}$$

{eq:y0}

$$y_0(\boldsymbol{\gamma}) = \exp(\boldsymbol{\gamma}^{\mathrm{T}}\mathbf{z}) \Rightarrow \ln y_0(\boldsymbol{\gamma}) = \boldsymbol{\gamma}^{\mathrm{T}}\mathbf{z} = \sum_{j=1}^{d} \gamma_j z_j. \tag{2.18}$$

{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 (2.12) and the survival function (2.14) into the log-likelihood (2.6), we get that the log-likelihood of a survival data set with the inverse gaussian FHT model, i.e.,

$$\begin{aligned} l(y_0, \mu, \sigma) = \sum_{i=1}^{n} \delta_i \Bigg(& \ln y_0 - \frac{1}{2}\ln\left(2\pi\sigma^2 t_i^3\right) - \frac{(y_0 + \mu t_i)^2}{2\sigma^2 t_i} \Bigg) \\ & + (1-\delta_i)\ln\Bigg(\Phi\left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}}\right) - \exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}}\right)\Bigg). \end{aligned} \tag{2.19}$$

{eq:loglik}

2.4.9 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 exist for doing this, and one of these for `R` (R Core Team, 2013) is 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 and setting

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 $\mathbf{X}$.

An important model class is the generalized additive model (GAM) (Hastie and Tibshirani, 1990). In a GAM, the conditional distribution of the response variable is assumed to follow an exponential family distribution. Then, the expected response $\mathbf{Y}|\mathbf{X}$ is modeled given the observed value $\mathbf{x}$ using a link function g as

$$g(\mathbb{E}(\mathbf{Y}|\mathbf{X} = \mathbf{x})) = f(\mathbf{x}), \tag{3.2}$$

meaning that if we use $g^{-1}(\cdot)$, the inverse of the link function, on this expression, we get

$$\mathbb{E}(\mathbf{Y}|\mathbf{X} = \mathbf{x})) = g^{-1}(f(\mathbf{x}). \tag{3.3}$$

This means that the conditional expectation of $\mathbf{Y}$ given $\mathbf{X}$ is a transformation of the additive predictor $f(\mathbf{x})$ using the inverse of the link function. We will not restrict ourselves to GAMs, but we will do something very similar. We will consider outcomes from a distribution

$$\psi(\mathbf{Y}|\mathbf{X} = \mathbf{x}, \theta), \tag{3.4}$$

which we will also at times refer to as a prediction function. We will model the conditional distribution of the parameter θ used in the prediction function, i.e.,

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

where $g(\cdot)$ is a chosen link function for the parameter θ. θ will typically correspond to the mean, but varying the distribution of course gives a lot of flexibility. In both these cases, we will call the function $f(\cdot)$ an additive predictor, which consists of the additive effects of the single predictors,

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

{eq:gam}

where β_0 is a common intercept and the functions $f_j(x_j), j = 1, \ldots, p$ are the partial effects of the variables x_j. The generic notation $f_j(x_j)$ may be different types of predictor effects such as classical linear effects $x_j\beta_j$, smooth non-linear effects constructed via regression splines, spatial effects or random effects of the explanatory variable x_j, and so on. In statistical boosting algorithms, we typically use component-wise effects, meaning that the different partial effects are estimated by separate base-learners $h_1(\cdot), \ldots, h_p(\cdot)$. Read more about this

in section 3.6 on component-wise boosting. The component-wise effects will be built up by additive estimation of base-learners.

We evaluate the 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 $f(\cdot)$. Very often, the loss ρ is derived from the negative log likelihood of the distribution of $\mathcal{Y}$. We denote this distribution $\psi(y)$. In case of GAMs, this will be the distribution of the corresponding exponential family. So the loss function is

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

Note that we will use the negative log likelihood as the loss function, due to the aim of *minimizing* the loss function, whereas the log likelihood increases as the model fits better to the data.

3.2.1 Example of a model and corresponding loss function

An example might be to model a normal linear regression. We assume data $D = (\mathbf{X}_i, Y_i)_{i=1}^N$ generated from a normal distribution $N(\mu, 1)$, where μ corresponds to θ above and is a functional which depends on covariates, i.e.,

$$\mu(\boldsymbol{\beta}, \mathbf{X}) = g(\boldsymbol{\beta}^T \mathbf{X}), \tag{3.8}$$

and where $g(\cdot)$ is a link function. For a continuous response $\mathbf{Y}$, we let the link function g be the identity link, such that

$$\mu(\boldsymbol{\beta}, \mathbf{X}) = \boldsymbol{\beta}^T \mathbf{X} = \beta_0 + \sum_{j=1}^{p} \beta_j x_j. \tag{3.9}$$

Although μ is a functional which depends on covariate matrix $\mathbf{X}$ and parameter vector $\boldsymbol{\beta}$, we will denote it μ whenever possible to lighten the notation. For a normally distributed observation, 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

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

which is the familiar L_2 loss function. Note that since we will only perform regression on μ, the loss function need not depend on σ^2, so we must only preserve the relevant proportionality.

This section needs to be rewritten.

3.2.2 Optimization

Obtaining the optimal prediction is usually accomplished by minimizing ρ. The goal of the model fitting scenario is to estimate a function which minimizes the

loss over an unseen "hold-out" sample, often called the out-of-sample error, the generalization error, or the test error. For a specific data set, we can calculate the empirical risk R, which is the sum of the loss function evaluated on all samples in the learning data set D,

$$R(D) = \sum_{i=1}^{n} \rho(y^{(i)}, f(x^{(i)})). \tag{3.11}$$

{eq:empirical-risk-2}

Other names for R are in-sample error and training error. 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,

$$\mathrm{Err} = \mathrm{E}[\mathrm{Err}_D] = \mathrm{E}_D\left[\rho(Y, f(\mathbf{X}))|D\right], \tag{3.12}$$

{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 (3.12), 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.

3.3 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 a statistical lens, and connected the successful machine learning approach to the world of statistical modelling. To understand gradient boosting, we first need to understand the gradient descent algorithm.

3.3.1 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

$$\nabla G(\mathbf{x}) = \left(\frac{\partial G(\mathbf{x})}{\partial x_1}, \frac{\partial G(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial G(\mathbf{x})}{\partial x_m} \right). \tag{3.13}$$

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

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 $\mathbf{x}_0$, e.g. $\mathbf{x}_0 = \mathbf{0}$. Let $m = 1$.
2. Calculate the direction to step in, $\mathbf{g}_{m-1} = -\nabla G(\mathbf{x}_{m-1})$.
3. Solve the line search to find the best step length a_m,
$$a_m = \operatorname*{argmin}_a \mathbf{x}_{m-1} + a\mathbf{g}_{m-1}.$$
4. The step in iteration m becomes $\mathbf{h}_m = a \cdot \mathbf{g}_{m-1}$.
5. Let $\mathbf{x}_m = \mathbf{x}_{m-1} + \mathbf{h}_{m-1}$.
6. Increase m, and go to step 2. Repeat until $m = M$.
7. The resulting minimum point is $\mathbf{x}_M = \mathbf{x}_0 + \sum_{m=1}^{M} \mathbf{h}_m(\mathbf{x}_m)$.

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.

3.3.2 Description of gradient boosting

Now, consider the problem of finding the additive predictor (3.6) which minimizes the empirical risk of a chosen loss function on a data set $D = \{x_i, y_i\}_{i=1}^N$:

$$\hat{f} = \underset{f}{\operatorname{argmin}}\, R(f) = \underset{f}{\operatorname{argmin}} \sum_{i=1}^{n} \rho(\mathbf{y}^{(i)}, f(\mathbf{x}^{(i)}). \tag{3.14}$$

The gradient boosting algorithm is gradient descent in the functional parameter space spanned by the base learners (Friedman, 2001). Friedman starts with suggesting a nonparametric approach. In this case, we consider each function value $f(x)$ to be a parameter, and then seek to minimize the empirical risk (3.11), as above. In function space, there are an infinite number of such parameters, since the space in which x exists is continuous. However, for our realized data set, there are only a finite number of such parameters, since we have N values of $\hat{f}(x_i), i = 1, \ldots, N$. We can then use the gradient descent algorithm as inspiration, and we take the solution $f(\cdot)$ to be a sum

$$f(\cdot) = f^{[0]}(\cdot) + \sum_{m=1}^{M} f^{[m]}(\cdot), \tag{3.15}$$

where the first $f^{[0]}(\cdot)$ is an initial guess, and the remaining $\{f^{[m]}(\cdot)\}_{m=1}^M$ are incremental functions – steps, or boosts – defined by the optimization method.

To use gradient descent on our objective function, the empirical risk, we need to compute its negative gradient, which we denote $\mathbf{u}$. There are two ways to arrive at it. One is to calculate the partial derivatives of the empirical risk (3.11), with respect to each estimated function value $\hat{f}^{[m-1]}(x_i)$:

$$\mathbf{u} = -\left(\frac{\partial}{\partial f(x_1)} R(f(\mathbf{x})), \ldots, \frac{\partial}{\partial f(x_n)} R(f(x_n))\right) \tag{3.16}$$

$$= -\left(\frac{\partial}{\partial f(x_1)} \sum_{i=1}^{N} \rho(y_i, f(x_i)), \ldots, \frac{\partial}{\partial f(x_n)} \sum_{i=1}^{N} \rho(y_i, f(x_i))\right) \tag{3.17}$$

$$= -\left(\frac{\partial}{\partial f(x_1)} \rho(y_1, f(x_1)), \ldots, \frac{\partial}{\partial f(x_n)} \rho(y_n, f(x_n))\right) \tag{3.18}$$

We see that each element in this vector $\mathbf{u}$ consists of the partial derivative of the loss function, with respect to each sample. However these partial derivatives are equal except for the y_i's and the $\hat{f}(x_i)$'s. In other words, we can simplify the notation as

$$\mathbf{u}^{[m-1]} = \left(-\frac{\partial}{\partial f} \rho(y_i, f(x_i))\big|_{f=\hat{f}^{[m-1]}}\right)_{i=1}^{N} \tag{3.19}$$

This $\mathbf{u}^{[m-1]}$ is a vector of *generalized residuals*. We could also have arrived at it by simply taking the derivative of the loss function $\rho(y, \hat{f}(x))$ with respect to $\hat{f}$, and made the vector by plugging in each sample.

With the generalized residuals in hand, we should now be able to minimize the loss function by performing gradient descent. However, this nonparametric approach of simply reducing the error of each data point will not generalize,

because we are only looking at the observed data points, and not at neighboring points in $\mathcal{X}$ space. We must therefore impose smoothness to neighboring points. But most importantly, we in any case wish to have an interpretable model. So we choose a base learner

$$h(\cdot), \tag{3.20}$$

like discussed previously. These base learners are usually relatively simple parametric effects 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. The reason we use simple models as base learners is that often algorithms exist for very fast computation of estimates, and there is also less to gain through combining complex learners.

We start with an initial guess $f_0(\cdot)$, say, a constant. Then we iterate, let us say, at each step m ($m > 0$) first calculating the generalized residuals from the previous iteration,

$$\mathbf{u}^{[m-1]} = \left(-\frac{\partial}{\partial \hat{f}(x)} \rho(y_i, \hat{f}(x)) \big|_{\hat{f}=\hat{f}^{[m-1]}} \right)_{i=1}^{N}, \tag{3.21}$$

where we insert the model from the previous step, $\hat{f}^{[m-1]}$. We wish to improve on these generalized residuals as much as possible, therefore we use a gradient descent step, but constrained to the base learner $h(\cdot)$. The function that we wish to choose, which minimizes the residuals, is the member of the base learner class that produces the $\hat{h}^{[m]}$ which is *most parallel* to $\mathbf{u}^{[m-1]}$, i.e. the h that is 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 basel earner. 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

$$\hat{\boldsymbol{\beta}}^{[m]} = \left(\left(\mathbf{u}^{[m-1]}\right)^T \mathbf{u}^{[m-1]} \right)^{-1} \left(\mathbf{u}^{[m-1]}\right)^T \mathbf{y}. \tag{3.22}$$

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,

$$a^{[m]} = \underset{a}{\operatorname{argmin}}\, R(\hat{f}^{[m-1]} + a^{[m]} \cdot \hat{h}^{[m]}\cdot. \tag{3.23}$$

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

$$f^{[m]}(\cdot) \leftarrow f^{[m-1]}(\cdot) + a^{[m]} \hat{h}^{[m]}(\cdot). \tag{3.24}$$

We iterate this procedure until some stopping criterion.

We are using gradient descent to find the parameters in each iteration, i.e., to find each . In other words, doing gradient descent in parameter space . So boosting can be viewed as an optimization procedure in functional space. For a schematic overview, see Algorithm 2.

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,

$$\hat{f} = \underset{f}{\operatorname{argmin}} R(f) = \underset{f}{\operatorname{argmin}} \sum_{i=1}^{n} \rho(y_i, f(x_i)). \tag{3.25}$$

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.,

$$\hat{f}_0(\cdot) = \beta_0(\cdot) = \underset{c}{\operatorname{argmin}} R(c). \tag{3.26}$$

3. Specify a base learner h.

4. Increase m by 1.

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

$$\mathbf{u}^{[m-1]} = \left(-\frac{\partial}{\partial f} \rho(y_i, f(x_i)) \big|_{f=\hat{f}^{[m-1]}} \right)_{i=1}^{N} \tag{3.27}$$

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:

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

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

3.3.3 Step length

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

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

3.3.4 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.

3.3.5 Practical considerations

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

3.4 likelihood-based boosting

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

3.5 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!!

3.5.1 L_2Boost example

Lorem ipsum.

3.6 High dimensions and component-wise gradient boosting

sec:component

In modern biomedical statistics, it is crucial to be able to handle high-dimensional data. In some situations, a data set consists of more predictors p than observations N. When p is much larger than N ($p >> N$), we talk about high-dimensional settings. In order to address the issue of analyzing high-dimensional data sets, a variety of regression techniques have been developed over the past years. Many of these techniques are characterized by a built-in mechanism for "regularization", which means that shrinkage of coefficient estimates or selection of relevant predictors is carried out simultaneously with the estimation of the model parameters. Both shrinkage and variable selection will typically improve prediction accuracy: In case of shrinkage, coefficient estimates tend to have a slightly increased bias but a decreased variance, while in case of variable selection, overfitting the data is avoided by selecting only the most informative predictors. For instance in the L_2Boost algorithm, if one uses a least squares base learner which uses all p dimensions, we see that it is infeasible: The matrix which must be inverted is singular when the number of 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.

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,

$$\hat{f} = \underset{f}{\operatorname{argmin}} R(f) = \underset{f}{\operatorname{argmin}} \sum_{i=1}^{n} y_i - \hat{f}(x_i). \tag{3.28}$$

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.,

$$f_0(\cdot) = \underset{c}{\operatorname{argmin}} R(c). \tag{3.29}$$

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

$$h(\mathbf{x}, \boldsymbol{\beta}) = \mathbf{x}^T \boldsymbol{\beta} = \sum_{j=1}^{p} \beta_j x_j \tag{3.30}$$

4. Increase m by 1.

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

$$\mathbf{u}^{[m-1]} = \left(y_i - \hat{f}(x_i)\right)_{i=1}^{N} \tag{3.31}$$

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

3.6.1 Stagewise, not stepwise

Component-wise gradient boosting is an algorithm which works very well in these settings. In fact, Buhlmann believes that it is mainly in the case of high-dimensional predictors that boosting has a substantial advantage over classical approaches (Bühlmann, 2006). The component-wise approach was first proposed in the L_2Boost paper (Bühlmann and Yu, 2003), and is very much an active field of research (Bühlmann, 2006; Mayr et al., 2014a,b, 2017). In the gradient boosting algorithm described in algorithm (2), we start out with an additive predictor $f^{[0]}(\cdot)$ which only consists of a constant. We have not added any effects of covariates yet. Instead of adding a small effect from all predictors, the component-wise approach 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.

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, one uses a set $\mathcal{H}$ of base learners, where all base learners are univariate. Typically these base learner 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

$$\mathcal{H} = \{h_1(\mathbf{x}; \beta_1) = \beta_1 x_1, h_2(\mathbf{x}; \beta_2) = \beta_2 x_2, \ldots, h_p(\mathbf{x}; \beta_p) = \beta_p x_p\}. \tag{3.32}$$

The initialization is the same as in the FGD algorithm. We first initialize the additive predictor $f^{[0]}(\cdot)$ to a constant β_0, and then specify a base learner. Now we specify a base learner for each component, as explained above. We then iterate. In an iteration m, we first calculate the generalized residuals by calculating the negative gradient where we insert the additive predictor from the previous step,

$$\mathbf{u}^{[m-1]} = \left(-\frac{\partial}{\partial f} \rho(y_i, f(x)) \big|_{f=\hat{f}^{[m-1]}} \right)_{i=1}^{N}. \tag{3.33}$$

Note that this is exactly like in the generic FGD 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

$$\hat{h}_1^{[m]}(\cdot), \hat{h}_2^{[m]}(\cdot), \ldots, \hat{h}_p^{[m]}(\cdot), \tag{3.34}$$

These estimated learners can 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. We select the best-fitting

base-learner $h^{[m]}_{j^{[m]}}$ based on the residual sum of squares of the base-learner fit

$$j^{[m]} = \underset{j \in \{1,2,\ldots,p\}}{\operatorname{argmin}} \sum_{i=1}^{N} \left(u_i - \hat{h}^{[m]}_j \right)^2 . \tag{3.35}$$

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

$$f^{[m]}(\cdot) \leftarrow f^{[m-1]}(\cdot) + \nu \cdot \hat{h}_{j^{[m]}}. \tag{3.36}$$

We iterate until the iteration number m becomes the pre-specified stopping iteration m_{stop}. The final model then becomes

$$\hat{f} = f^{[m_{\text{stop}}]}(\cdot) = \beta_0 + \sum_{m=1}^{m_{\text{stop}}} \nu \cdot \hat{h}^{[m]}_{j^{[m]}}(\cdot). \tag{3.37}$$

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.,

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

hence the resulting additive predictor is a sum of component-wise predictors in the GAM form of

$$\hat{f}(\mathbf{x}) = \beta_0 + \sum_{j=1}^{p} \hat{f}_j(x_j). \tag{3.38}$$

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

3.7 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.

3.8 Selecting m_{stop}

As we mentioned in subsection 3.3.4, the crucial tuning parameter in boosting is the number of iterations, m_{stop}. Stopping early enough performs variable selection and shrinks the parameter estimates toward zero. In the case of

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,

$$\hat{f} = \underset{f}{\operatorname{argmin}} R(f). \tag{3.39}$$

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.,

$$\beta_0 = \underset{c}{\operatorname{argmin}} R(c). \tag{3.40}$$

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,

$$\mathbf{u}^{[m-1]} = \left(-\frac{\partial}{\partial f} \rho(y_i, f(x_i)) \big|_{f=\hat{f}^{[m-1]}} \right)_{i=1}^{N}. \tag{3.41}$$

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

$$\hat{h}_1^{[m]}(\cdot), \hat{h}_2^{[m]}(\cdot), \ldots, \hat{h}_p^{[m]}(\cdot). \tag{3.42}$$

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

$$j^{[m]} = \underset{j \in \{1,2,\ldots,p\}}{\operatorname{argmin}} \sum_{i=1}^{N} \left(u_i - \hat{h}_j^{[m]} \right)^2. \tag{3.43}$$

last-step

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

$$\hat{f}^{[m]}(\cdot) \leftarrow \hat{f}^{[m-1]}(\cdot) + \nu \cdot \hat{h}_{j^{[m]}}^{[m]}(\cdot). \tag{3.44}$$

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

10. Return the final boosted additive predictor

$$\hat{f}(\cdot) = \hat{f}^{[m_{\text{stop}}]}(\cdot) = \beta_0 + \sum_{m=1}^{m_{\text{stop}}} \nu \cdot \hat{h}_{j^{[m]}}^{[m]}(\cdot) \tag{3.45}$$

$p < N$, 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?

3.8.1 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

3.8.2 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

$$\text{CV}(m) = \sum_{k=1}^{K} \sum_{i \in \kappa(k)} \rho(y_i, \hat{y}_i^{-\kappa(k)}). \tag{3.46}$$

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,

$$m_{\text{stop}} = \underset{m}{\text{argmin}}\, \text{CV}(m). \tag{3.47}$$

Typical values for K are 5 or 10, but in theory one can choose any number. The extreme case is $K = N$, called leave-one-out cross-validation, where all but one

observation is used for training and the model is evaluated on the observation that was left out. In this case, the outcome is deterministic, since there is no randomness when dividing into folds.

3.8.3 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).

3.8.4 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,

$$\text{RCV}(m) = \sum_{j=1}^{J} \sum_{k=1}^{K} \sum_{i \in \kappa(j,k)} \rho(y_i, \hat{y}_i^{-\kappa(j,k)}). \tag{3.48}$$

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

$$m_{\text{stop}} = \underset{m}{\text{argmin}}\, \text{RCV}(m). \tag{3.49}$$

In practice, to ensure we find the minimizing m, we let the boosting algorithm run for $m = 1$ to $m = M$, where M is a large number that we are sure will result in a overfitted model.

3.9 Multidimensional boosting: Cyclical component-wise

A limitation of all the classical boosting methods we have described earlier, as well as of L_1-penalized estimation such as the lasso method (Tibshirani, 1996), is that they are designed for statistical problems involving a one-dimensional prediction function. By only considering such functions, we are restricted to

estimating models which only model a single quantity of interest, which is almost always the mean. In many applications, modelling only one parameter will not be sufficient. 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 2.4, 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 an example of the multidimensional boosting algorithm first introduced in Schmid et al. (2010). We will here explain the algorithm used in this paper.

Schmid et al. (2010) extend the component-wise gradient boosting algorithm (Friedman, 2001) to such a setting where one has a prediction function ψ, typically a probability distribution function (pdf), which has K parameters,

$$\psi(\mathbf{Y}|\mathbf{X}=\mathbf{x}) = \psi(\mathbf{Y}|\boldsymbol{\theta}(\mathbf{x})) = \psi(\mathbf{Y}|\theta_1(\mathbf{x}), \theta_2(\mathbf{x}), \ldots, \theta_K(\mathbf{x})). \tag{3.50}$$

We will model each parameter $\theta_k, k = 1, \ldots, K$ with its own additive predictor, meaning modelling its conditional mean as

$$\mathbb{E}(\theta_k|\mathbf{X}=\mathbf{x}) = g_k^{-1}(f_k(\mathbf{x})), \tag{3.51}$$

where $g_k^{-1}(\cdot)$ is the inverse of a chosen link function, which must then be required to be invertible, for parameter k. Furthermore, $f_k(\cdot)$ is an additive predictor

$$f_k(\mathbf{x}) = \beta_{k,0} + f_{k,1}(x_1) + f_{k,2}(x_2) + \ldots + f_{k,p}(x_p), \tag{3.52}$$

where each component $x_j, j = 1, \ldots, p$ in the covariate matrix $\mathbf{X}$ has its own predictor $f_{k,j}(x_j)$. The link functions are typically used to constrain the domain of the parameter. For example, if we use the logarithm as the link function, the inverse link function is the exponential function, meaning that

$$\mathbb{E}(\theta_k|\mathbf{X}=\mathbf{x}) = \exp(f_k(\mathbf{x})),$$

which will constrain the expectation to be a positive number.

We may consider θ_k a functional which takes the covariate matrix $\mathbf{X}$. We denote the full set of all additive predictors

$$\mathbf{f}(\cdot) = (f_1(\cdot), f_2(\cdot), \ldots, f_K(\cdot)). \tag{3.53}$$

To evaluate the fit of the additive predictors, we use a loss function $\rho(y, \boldsymbol{\theta}(\mathbf{x}))$, which as usual typically is the negative log likelihood of the distribution of ψ. Like in regular statistical boosting, we must be able to obtain derivatives of the loss function with regard to the parameters $\boldsymbol{\theta}$. Therefore, it is assumed that all partial derivatives

$$\frac{\partial}{\theta_1}\rho(\boldsymbol{\theta}), \frac{\partial}{\theta_2}\rho(\boldsymbol{\theta}), \ldots, \frac{\partial}{\theta_K}\rho(\boldsymbol{\theta}) \tag{3.54}$$

exist. The main idea of the cyclical multidimensional boosting algorithm (although cyclical was a term coined later by Thomas et al. (2018)) is to have a boosting step for each parameter k, in each iteration. In other words, we cycle through all parameter dimensions in each boosting iteration. The initialization of the algorithm is done analogously to the regular boosting method, by setting each parameter to a constant, typically zero or the constant which maximizes the likelihood of the data. We also specify a base learner for each parameter dimension k. 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 be the same as in the generic FGD algorithm (2), i.e., estimating one full base learner, or it can be to estimate a base learner for each covariate, and then select the best-fitting one, like in the component-wise gradient boosting algorithm (4). The latter approach is typically used, since it has been shown that component-wise boosting algorithms often are powerful. In iteration m, after having cycled through to component k, the estimated parameter vector is

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

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. We calculate a residual for dimension k by calculating the k-th partial derivative and inserting the current estimated parameter vector $\hat{\boldsymbol{\theta}}_{k-1}^{[m]}$, and evaluating it at the observations $x_1, x_2, \ldots, x_N$. This yields the generalized residual vector

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

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 $j_k^{[m]}$ and corresponding estimated learner $\hat{h}_{j^{[m]}}^{[m-1]}(\cdot)$, and update the parameter in dimension k by the usual

$$\theta_k^{[m]} \leftarrow \theta_k^{[m-1]} + \nu \cdot \hat{h}_{j_k^{[m]}}^{[m]}(\cdot). \tag{3.55}$$

A schematic representation of the updating process using this algorithm in a

given iteration m looks as follows:

$$\frac{\partial}{\theta_1}\rho\left(\hat{\theta}_1^{[m-1]}, \hat{\theta}_2^{[m-1]}, \hat{\theta}_3^{[m-1]}, \ldots, \hat{\theta}_{K-1}^{[m-1]}, \hat{\theta}_K^{[m-1]}\right) \xrightarrow{\text{calculate}} \mathbf{u}_1^{[m-1]} \xrightarrow{\text{fit and update}} \hat{\theta}_1^{[m]}$$

$$\frac{\partial}{\theta_2}\rho\left(\hat{\theta}_1^{[m]}, \hat{\theta}_2^{[m-1]}, \hat{\theta}_3^{[m-1]}, \ldots, \hat{\theta}_{K-1}^{[m-1]}, \hat{\theta}_K^{[m-1]}\right) \xrightarrow{\text{calculate}} \mathbf{u}_2^{[m-1]} \xrightarrow{\text{fit and update}} \hat{\theta}_2^{[m]}$$

$$\frac{\partial}{\theta_3}\rho\left(\hat{\theta}_1^{[m]}, \hat{\theta}_2^{[m]}, \hat{\theta}_3^{[m-1]}, \ldots, \hat{\theta}_{K-1}^{[m-1]}, \hat{\theta}_K^{[m-1]}\right) \xrightarrow{\text{calculate}} \mathbf{u}_3^{[m-1]} \xrightarrow{\text{fit and update}} \hat{\theta}_3^{[m]}$$

$$\ldots$$

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

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

After cycling through all estimation parameters in ρ, the algorithm does an additional step to find the current optimal nuisance parameters. If indeed the loss function has any nuisance parameters. A nuisance parameter is any parameter which is not of immediate interest but which must be accounted for in the analysis of those parameters which are of interest. For example, if we are considering a multivariate normal distribution and are only interested in the means, the residual variance would be a nuisance parameter. We find the optimal nuisance parameters by performing numerical minimization of the empirical risk for one scale parameter at a time. For each nuisance parameter $\sigma_l, l = 1, 2, \ldots, L$, we plug all other estimated parameters into the empirical risk, and minimize it over the current nuisance parameter σ_l:

$$\hat{\sigma_l}^{[m]} = \underset{\sigma_l}{\operatorname{argmin}} \sum_{i=1}^{N} \rho(y_i, \hat{f}^{[m]}, \hat{\sigma}_1^{[m]}, \hat{\sigma}_2^{[m]}, \ldots, \hat{\sigma}_{l-1}^{[m]}, \sigma_l, \hat{\sigma}_{l+1}^{[m-1]}, \ldots, \hat{\sigma}_L^{[m-1]}) \tag{3.56}$$

The gamboostLSS algorithm (Mayr et al., 2012a) does not include this step because GAMLSS do not have any nuisance parameters. For a schematic overview of the algorithm, see (5). Note that this algorithm resembles the backfitting strategy by Hastie and Tibshirani (1986). In both strategies, 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, but in gradient boosting, 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, Schmid et al. (2010) say that it should not run until convergence, but rather find estimates by cross-validation. An issue with this algorithm, though, is that for proper tuning it requires a vector $\mathbf{m}_{\text{stop}}$ of stopping iterations, one 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.

Algorithm 5 Multidimensional cyclical component-wise gradient boosting

algo:multi-cyclical

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

$$\hat{\mathbf{f}} = \underset{\mathbf{f}}{\operatorname{argmin}} R(\mathbf{f}). \tag{3.57}$$

2. Initialize iteration counter m to 0. Initialize additive predictors $f_1^{[0]}, f_2^{[0]}, \ldots, f_K^{[0]},$, to $\beta_{1,0}, \beta_{2,0}, \ldots, \beta_{K,0}$, respectively. This can be done e.g., by setting them to zero for all components, or by finding the best constant by a maximizing the likelihood of the data. If there are nuisance parameters $\sigma = \sigma_1, \ldots, \sigma_L$, initialize these as well, $\sigma_l^{[0]}, l = 1, \ldots, L$.

initialization

3. Specify a set of base learners $\mathcal{H}_k$ for each dimension $k = 1, \ldots, K$. 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 10. Otherwise compute the negative partial derivative $-\frac{\partial \rho}{\partial \hat{f}_k}$ and evaluate at $\hat{\mathbf{f}}^{[m-1]}(x_i), i = 1, \ldots, N$, yielding the negative gradient vector

$$\mathbf{u}_k^{[m-1]} = \left(-\frac{\partial}{\partial f_k} \rho(y_i, \mathbf{f}(x_i)) \big|_{\mathbf{f}=\hat{\mathbf{f}}^{[m-1]}} \right)_{i=1}^N \tag{3.58}$$

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,

$$\hat{h}_{k,1}^{[m]}, \hat{h}_{k,2}^{[m]}, \ldots, \hat{h}_{k,p}^{[m]} \tag{3.59}$$

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 best fits $\mathbf{u}_k^{(m-1)}$ according to RSS, or, rather, the best-fitting base learner,

$$j_k^{[m]} = \underset{j \in \{1,2,\ldots,p\}}{\operatorname{argmin}} \sum_{i=1}^N \left(u_{k,i} - \hat{h}_{k,j}^{[m]} \right)^2. \tag{3.60}$$

cyclic-last

9. Update the additive predictor for component k by

$$\hat{f}_k[m] \leftarrow \hat{f}_k^{[m-1]} + \nu \cdot \hat{h}_{j_k^{[m]}}^{[m]}, \tag{3.61}$$

where ν is the real-valued step-length factor specified in step 3. Then update the full model thus far.

repeat-step

10. If $k = K$, go to step 11. If not, go to step 6.

nuisance-init

11. Set $l = 0$.

nuisance-first

12. Increase l by 1.

13. Plug $\hat{f}^{(m)}$ and $\hat{\sigma}_1^{[m-1]}, \ldots, \hat{\sigma}_{l-1}^{[m-1]}, \hat{\sigma}_{l+1}^{[m-1]}, \hat{\sigma}_L^{[m-1]}$ into the empirical risk function R and minimize the empirical risk over σ_l. Set $\hat{\sigma}_l^{[m]}$ equal that minimizer.

14. Repeat steps 13 and 14 for $l = 2, \ldots, L$.

grid-search

3.9.1 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 finite grid with N_k points, such that we obtain

$$m_{\min,k} = m_{1,k} < m_{2,k} < \ldots < m_{N_k-1,k} < m_{N_k,k} = m_{\max,k}, \tag{3.62}$$

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 3.8.2, 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 3.8.2, 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.

3.10 Noncyclical component-wise multidimensional boosting algorithm

In the cyclical algorithm seen previously, 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 3.9.1, 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).

3.10.1 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,

$$\hat{h}_{\theta_k}(\cdot), \tag{3.63}$$

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

$$f_{\theta_k}^{[m+1]} = f^{[m]} + \nu \cdot \hat{h}_{\theta_k}(\cdot). \tag{3.64}$$

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

$$R(f_{\theta_k}^{[m+1]}). \tag{3.65}$$

We calculate the gain in the loss function, which we denote $\Delta\rho_{\theta_k}$, by

$$\Delta\rho_{\theta_k} = R(f^{[m]}) - R(f_{\theta_k}^{[m+1]}). \tag{3.66}$$

If we now compare the gain in loss function value, ρ_{θ_k}, across each parameter $k = 1, 2, \ldots, K$, we can find out which parameter leads to the best increase. We choose that one, i.e.,

$$k^* = \underset{k}{\operatorname{argmin}}\, \Delta\rho_{\theta_k}, \tag{3.67}$$

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

$$f^{[m+1]} \leftarrow f_{\theta_k}^{[m+1]} = f^{[m]} + \nu \cdot \hat{h}_{\theta_k}(\cdot). \tag{3.68}$$

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.,

$$j^* = \underset{j}{\operatorname{argmin}} \sum_{i=1}^{N} (u_{k,i} - (f^{[m]}(x_i) + \hat{h}(\mathbf{x}_i)))^2. \tag{3.69}$$

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.,

$$j^* = \underset{j}{\operatorname{argmin}} \sum_{i=1}^{N} \rho(y_i, f(x_i)). \tag{3.70}$$

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, $\Delta\rho_{\theta_k}$, 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,
$$\hat{f} = \underset{f}{\operatorname{argmin}} R(f). \tag{3.71}$$
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
$$\mathbf{u}_k^{(m-1)} = \left(-\frac{\partial}{\partial \hat{f}_k} \rho(y_i, \hat{f}^{(m-1)}(x_i)) \right)_{i=1}^N \tag{3.72}$$
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
   $$j^* = \underset{j \in 1, \ldots, J_k}{\operatorname{argmin}} \sum_{i=1}^N (u_k^{(i)} - \hat{h}_{kj}(x^{(i)}))^2 \tag{3.73}$$
   - the outer loss, i.e., the loss function after the potential update,
   $$j^* = \underset{j \in 1, \ldots, J_k}{\operatorname{argmin}} \sum_{i=1}^N \rho \left(y^{(i)}, \hat{f}^{(m-1)}(x^{(i)}) + \nu \cdot \hat{h}_{kj}(x^{(i)}) \right) \tag{3.74}$$
10. Compute the possible improvement of this update regarding the outer loss,
$$\Delta\rho_k = \sum_{i=1}^N \rho \left(y^{(i)}, \hat{f}^{(m-1)}(x^{(i)}) + \nu \cdot \hat{h}_{kj^*}(x^{(i)}) \right) \tag{3.75}$$
11. Update, depending on the value of the loss reduction, $k^* = \operatorname{argmin}_{k \in 1, \ldots, K} \Delta\rho_k$
$$\hat{f}_{k^*}^{(m)} = \hat{f}_{k^*}^{(m-1)} + \nu \cdot \hat{h}_{k^* j^*}(x), \tag{3.76}$$
while for all $k \neq k^*$,
$$\hat{f}_{k^*}^{(m)} = \hat{f}_{k^*}^{(m-1)}. \tag{3.77}$$
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 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.

4.2 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.

4.2.1 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.

4.2.2 Derivatives not on same scale

Pass.

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 $\text{IG}(\mu, y_0)$.
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 $(\widetilde{t}_i, \delta_i)_{i=1,\ldots,N}$.

4.2.3 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.

4.3 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.

4.4 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 (4.4) 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

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.

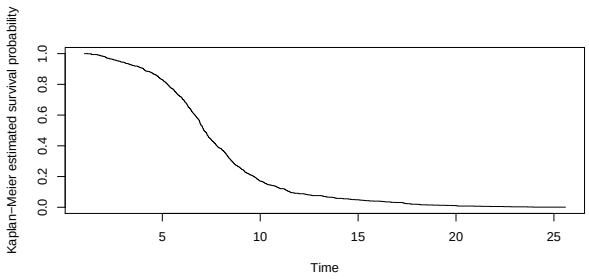
   h) Update $\hat{\mu}^{[m-1]} \leftarrow \hat{\mu}^{[m-1]} + \nu \hat{U}_{\mu}^{[m-1]}$.

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

Figure 4.1: Kaplan-Meier plot of small example



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.

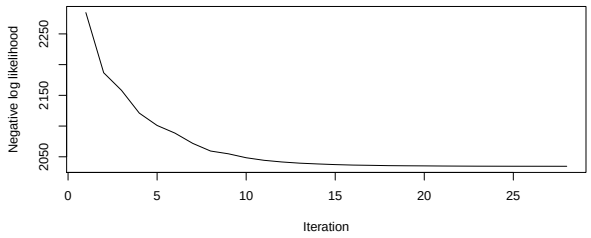
4.4.1 Small example

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 data using Algorithm 4.4, 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.

Figure 4.2: Log-likelihood



4.5 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.

4.5.1 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).

4.5.2 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.

4.6 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 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

table:non-correlated-with-intercept-summary

Table 4.1: Summary of results

Measure	Mean	Standard deviation	Minimum	Maximum
Deviance	95.371	76.742	-7.2	1418.1
m_{stop}	15.9	6.4	2	39

table:non-correlated-with-intercept-y0

Table 4.2: Result for y_0, β

Measure	Mean	Standard deviation
Sensitivity	0.188	0.092
Specificity	1.000	0.000
Accuracy	0.997	0.000
FPR	0.000	0.000
FNR	0.812	0.092
FOR	0.003	0.000
NPV	0.997	0.000

table:non-correlated-with-intercept-mu

Table 4.3: Result for μ, γ

Measure	Mean	Standard deviation
Sensitivity	0.730	0.248
Specificity	0.944	0.109
Accuracy	0.872	0.091
FPR	0.056	0.109
FNR	0.270	0.248
FOR	0.112	0.093
NPV	0.888	0.093

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.

4.6.1 Variable selection results

We here try to summarize how the model selects the informative variables.

4.6.2 Boosting with changing the intercept

See Table 4.1, Table 4.2, and 4.3.

4.6.3 Boosting *without* changing the intercept

See Table 4.4, Table 4.5, and 4.6.

Table 4.4: Summary of results

table:non-correlated-no-intercept-summary

Measure	Mean	Standard deviation	Minimum	Maximum
Deviance	129.779	41.207	-8.0	255.2
m_{stop}	63.7	26.6	2	160

Table 4.5: Result for y_0, β

table:non-correlated-no-intercept-y0

Measure	Mean	Standard deviation
Sensitivity	0.452	0.163
Specificity	0.997	0.002
Accuracy	0.995	0.001
False positive rate	0.003	0.002
False negative rate	0.548	0.163
False omission rate	0.002	0.001
Negative predictive value	0.998	0.001

Table 4.6: Result for μ, γ

table:non-correlated-no-intercept-mu

Measure	Mean	Standard deviation
Sensitivity	0.953	0.120
Specificity	0.640	0.291
Accuracy	0.744	0.186
False positive rate	0.360	0.291
False negative rate	0.047	0.120
False omission rate	NA	NA
Negative predictive value	NA	NA

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-loglik}

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-loglik-proper}

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

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

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

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

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.

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.

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

saddlepoints
Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2933–2941, Cambridge, MA, USA. MIT Press.

DeBin2016
De Bin, R. (2016). Boosting in cox regression: a comparison between the likelihood-based and the model-based approaches with focus on the r-packages coxboost and mboost. *Computational Statistics*, 31(2):513–531.

eaton-whitmore
Eaton, W. W. and Whitmore, G. A. (1977). Length of stay as a stochastic process: A general approach and application to hospitalization for schizophrenia. *The Journal of Mathematical Sociology*, 5(2):273–292.

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.

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.

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.

whitmore1979 Whitmore, G. A. (1979). An inverse gaussian model for labour turnover. *Journal of the Royal Statistical Society. Series A (General)*, 142(4):468–478.

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.