Boosting the First-Hitting-Time Regression Model

Vegard Stikbakke

February 2, 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

Survival analysis is the field of studying lifetime and time-to-event data. An overview of modelling survival data is Aalen et al. (2008). We look at a stochastic variable $T > 0$ which is the time to 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 they 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.

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 $f(t)$.

The hazard function $\alpha(t)$

We are also interested in the hazard function. This is the probability of the event happening in a given small interval at time t, conditioned on the event not having happened yet. More formally, the hazard function is defined as a limit of this probability as the size of the interval goes to zero,

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

The hazard function $\alpha(t)$ is then the chance of the event happening at time t, if it has not happened yet. Estimation of the hazard function is hard, and we do not achieve the usual $\sqrt{n}$ convergence.

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 rate 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

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

and the cumulative hazard function $A(t)$ by the Nelson-Aalen estimator

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

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 regression coefficient, we want to set up a likelihood for the data. We assume that the data is independent and identically distributed. If the event has occurred, the indicator d_i is 1. 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. If the event has occurred, d_i is 1, and then $1 - d_i$ is 0. Similarly, if the event has not occurred and the event is censored, then d_i is 0, and then $1 - d_i$ is 1. This allows us to take the product of (2.3) and (2.4) where we take these to the power of δ_i and $1 - \delta_i$, respectively, so 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 of the observed sample as a whole is the product of the single likelihoods. 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 derive 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 $\log S(t) = -A(t)$ (2.2) and $f(t) = \alpha(t)S(t)$ (2.1), this 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

So far we have not introduced covariates. 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,

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

{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 coefficient $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)$. We choose $r(\mathbf{x})$ such that it is appropriately normalized, meaning $r(\mathbf{0}) = 1$. A vital assumption here is that the covariates are fixed in time. With 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,

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

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 the Cox model (Cox, 1992),

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

This 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 a tremendous amount in applied research.

Cox regression example

Lorem ipsum.

The proportional hazards assumption

When we say (2.7), that $\alpha(t|\mathbf{x}) = \alpha_0(t) r(x|\boldsymbol{\beta})$, we make the proportional hazards (PH) assumption: We assume that the ratio between the hazard function of two individuals is the same *at all times*. This is a very large assumption to make, and in practice, it is very often not the case. One way to test this assumption for a covariate $j = 1, \ldots, p$, is to fit a model $r(x_j) = \exp(f(x_j))$, where $f(\cdot)$ is some spline regression function, and plot it against x_j.

Robustness of Cox when the PH assumption is violated

CITATION NEEDED

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

2.4 First hitting time models or threshold regression

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 it takes for 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

$$T = \underset{t}{\operatorname{argmin}}\, Y(t) \in \mathcal{B}.$$

First hitting time (FHT) models were introduced in Whitmore (1986), and a good reference paper on the topic is Lee and Whitmore (2006). Note that these authors use the term threshold regression. We have, together with Caroni (2017), chosen to not use this term, as it is also the name of an already established, and quite different, field of econometrics. FHT models have been applied to many different fields, including medicine, engineering, and economics. They may describe the survival time of a transplant patient, the duration time of a strike, the failure time of an engineering system, and so on.

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 turns out that it yields a fully parametric regression model.

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

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

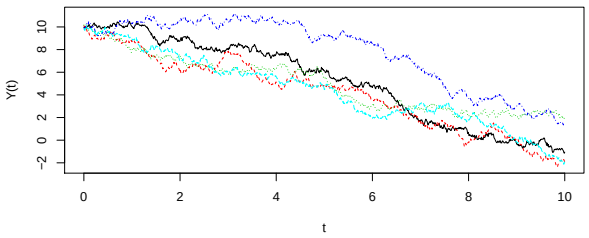
we call W a Wiener process. In other words, each increment has expectation 0 and has standard deviation proportional to the length of the time interval. The position of the process at time t always follows a Gaussian distribution $N(0, t)$ (Aalen et al., 2008). To increase the flexibility of the Wiener process, we can introduce a new process Y,

$$Y(t) = y_0 - \mu t + \sigma W(t), \tag{2.8}$$

{wiener}

which is called a Wiener process with initial value y_0, drift coefficient μ, and diffusion coefficient σ. Introductions to many aspects of Wiener processes are found in Cox and Miller (1965). Figure 2.4 shows simulations of 5 Wiener process paths with initial value $y_0 = 10$ and negative drift $\mu = 1$.

plot:wiener



FHT with Wiener process leads to Inverse Gaussian

If we choose the stochastic process to be a Wiener process like in (2.8), and we let the boundary be the non-positive numbers, $\mathcal{B} = (-\infty, 0]$, then the lifetime is the time it takes for the process to first reach a non-positive value,

$$T = \underset{t}{\operatorname{argmin}}\, Y(t) \leq 0. \tag{2.9}$$

Note that since the Wiener process is continuous, there will not be a difference between $\leq$ and $<$.

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 do want a monotonic restriction on the movement of the process, we may use a gamma process (Lee and Whitmore, 2006).

It can be shown that the first hitting time of the Wiener process follows an inverse Gaussian distribution (Chhikara, 1988), with probability distribution function

$$f(t|y_0, \mu, \sigma^2) = \frac{y_0}{\sqrt{2\pi\sigma^2 t^3}} \exp\left[-\frac{(y_0 + \mu t)^2}{2\sigma^2 t}\right], \tag{2.10}$$

{eq:ig-pdf}

and cumulative distribution function

$$F(t|\mu, \sigma^2, y_0) = \Phi\left[-\frac{(\mu t + y_0)}{\sqrt{\sigma^2 t}}\right] + \exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left[\frac{\mu t - y_0}{\sqrt{\sigma^2 t}}\right]. \tag{2.11}$$

{eq:ig-cdf}

TO DO!

See Appendix for the mathematical derivation. 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 (2.10) is improper. In this case, the probability of the time not being finite is

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

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

{eq:ig-surv}

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

$$\Phi(x) = \int_{-\infty}^{x} \exp\left(\frac{-y^2}{2}\right) / \sqrt{2\pi}\, \mathrm{d}y, \tag{2.13}$$

and in (2.12) we used the fact that $1 - \Phi(-x) = \Phi(x)$, since the standard normal distribution is symmetric around 0.

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 (2.10) and

the survival function $S(t|\mu, \sigma^2, y_0)$ in (2.12) 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 do it (Lee and Whitmore, 2006).

The shape of the hazard rate

As previosly stated, The hazard rate is obtained from $\alpha(t) = f(t)/S(t)$ (2.1). Regardless of 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).

Comparison of hazard rates

Of particular interest might be looking at the ratio between two hazard rates, that is, one hazard divided by the other. 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. 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

$$\lim_{t\to\infty} \alpha(t) = \frac{1}{2}\left(\frac{\mu}{\sigma}\right)^2 = 0.5\mu^2 \tag{2.14}$$

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.

Regression

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.

$$\begin{aligned} \mu &= \boldsymbol{\beta}^{\mathrm{T}}\mathbf{x} \\ \ln y_0 &= \boldsymbol{\gamma}^{\mathrm{T}}\mathbf{z} \end{aligned} \tag{2.15}$$

{eq:coeffs}

Here $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ 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 (2.10) and the survival function (2.12) into the log-likelihood (2.6), we get that the log-likelihood of a survival data set with

the inverse gaussian FHT model is

$$l(y_0, \mu, \sigma) = \sum_{i=1}^{n} \delta_i \left(\ln y_0 - \frac{1}{2} \ln\left(2\pi\sigma^2 t_i^3\right) - \frac{\left(y_0 + \mu t_i\right)^2}{2\sigma^2 t_i} \right) + (1 - \delta_i) \ln\left(\Phi\left(\frac{\mu t_i + y_0}{\sqrt{\sigma^2 t_i}} \right) - \exp\left(-\frac{2 y_0 \mu}{\sigma^2} \right) \Phi\left(\frac{\mu t_i - y_0}{\sqrt{\sigma^2 t_i}} \right) \right). \tag{2.16}$$

{eq:loglik}

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. **This is the main focus of my thesis.**

Example of application

Lorem ipsum some example. Just use numerical maximization.

Identification problems

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., 2014). 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 99% 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 another overview, consult also the literature review article by Mayr et al. (2014).

3.1 AdaBoost

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 $(\mathbf{x}_i, y_i)_{i=1,\dots,n}$, where $x \in \mathbb{R}^p$ and $y \in \{-1, 1\}$, i.e., positive or negative; yes or no. We want to find a rule which best separates these observations into the correct buckets, 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. Other observations, however, 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, which initially is a plain sum of the observations. After this initial classification, some points will be correctly classified, and some will be misclassified. What we now do is we change the weights of the observations. 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, make a new 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. This was especially true in the early years of boosting. 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.

See some figure for a schematic overview of the algorithm.

3.2 Statistical model fitting

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 regression scheme. Let

$D = \{x^{(i)}, y^{(i)}\}_{i=1,\ldots,n}$ be a learning data set. We assume that the samples $i = 1, \ldots, n$ are sampled independently from an identical distribution over the joint space $\mathcal{X} \times \mathcal{Y}$. The input space 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 and continuous, e.g., in the standard regression setting. (In the censored survival data setting, however, it is in a sense two-dimensional, since we have $y^{(i)} = (t_i, d_i)$. But not really, since we are only outputting the time.)

We assume there exists some structure in the data to be found. We choose a model $f(\cdot)$ to model that structure. We can then produce predicted values $\hat{y}^{(i)} = f(x^{(i)})$. To calculate the difference between the predicted outcome and the actual outcome, we need a loss function ρ. It measures the difference, or distance, between the true outcome $y^{(i)}$ and the predicted outcome $\hat{y}^{(i)}$. A loss function must be symmetric and convex. Examples of ρ are the absolute loss $|y - \eta(x)|$, which leads to a regression model for the median, and the quadratic loss (the L_2 loss), which leads to the usual regression model for the mean. Very often, the loss is derived from the **negative** log likelihood of the distribution of $\mathcal{Y}$, depending on the desired model. Keep in mind that the negative is used, due to the aim of *minimizing* the loss function. In the survival data setting, typical loss functions are ROC and Briar score (Bøvelstad and Borgan, 2011).

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. (Or, in a statistical setting, one might use more traditional model selection criteria.) 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,

$$R(D) = \sum_{i=1}^{n} \rho(y^{(i)}, x^{(i)}). \tag{3.1}$$

{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,

$$\mathrm{Err} = \mathrm{E}[\mathrm{Err}_D] = \mathrm{E}_D\left[\rho(Y, f(\mathbf{X}))|D\right], \tag{3.2}$$

{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 (3.2), 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. 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

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

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.

3.3 Gradient boosting

Friedman (2001) developed an algorithm for fitting an additive model called gradient boosting. He showed that AdaBoost performs this algorithm, for a particular exponential loss function. See Hastie et al. (2009) for a good demonstration of the 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 the algorithm, we first need to understand the gradient descent algorithm.

Gradient descent

Suppose you 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

$$\nabla G(\mathbf{x}) = \left(\frac{\partial G(\mathbf{x})}{\partial x_1}, \frac{\partial G(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial G(\mathbf{x})}{\partial x_m}\right). \tag{3.3}$$

The motivation behind the gradient descent algorithm is that in a small interval around a point $\mathbf{x}_0 \in \mathbb{R}^m$, G is decreasing the most in the direction of the negative gradient at that point. Therefore, by taking a small step slightly in the direction of the negative gradient, from $\mathbf{x}_0$ to a new value $\mathbf{x}_1$, we end up with a slightly lower function value: The new function value $G(\mathbf{x}_1)$ will be less than $G(\mathbf{x}_0)$. In some versions of the algorithm, the step length $\nu \in (0, 1]$ is found by a line search, i.e., by finding the step length which gives the best improvement. In other versions, one simply uses a fixed step length. The gradient descent algorithm repeats this procedure until convergence. Indeed, with a sufficiently small step length, gradient descent will always converge, albeit possibly to a local minimum. For a schematic overview of the algorithm, see Algorithm 1. The gradient descent algorithm is surprisingly robust. Even though it may converge to a local minimum, it often seems to find good solutions globally. This is likely related to research which has found that in high-dimensional spaces, most minima are not minima, but in fact, saddlepoints masquerading as local minima (Dauphin et al., 2014). This means that training will slow since the gradient will be small at this saddlepoint or plateau. When using a gradient descent method typically one sets a threshold at which the algorithm terminates

algo:grad-desc

Algorithm 1 Gradient descent

We want to minimize $G(x)$, i.e. solve $\min_x G(x)$.

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,
$$a_m = \underset{a}{\operatorname{argmin}}\, \mathbf{x}_{m-1} + a\mathbf{g}_{m-1}.$$
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 minumum point is $\mathbf{x}_M = \mathbf{x}_0 + \sum_{m=1}^{M} \mathbf{h}_m(\mathbf{x}_m)$.

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.

Main idea of gradient boosting

Now, consider the problem of finding a model $f(\cdot)$ which minimizes the empirical risk of a chosen loss function (3.1) on a data set $D = \{x_i, y_i\}_{i=1}^{N}$,

$$\underset{f}{\operatorname{argmin}}\, R(f) = \underset{f}{\operatorname{argmin}} \sum_{i=1}^{n} \rho(y^{(i)}, f(x^{(i)})). \tag{3.4}$$

Friedman starts off with suggesting one might take 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 (3.1), as above. In function space, there are infinite 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. We can then use the gradient descent algorithm as inspiration, and we take the solution $f(\cdot)$ to be a sum

$$f(\cdot) = \sum_{m=0}^{M} f_m(\cdot), \tag{3.5}$$

where the first $f_0(\cdot)$ is an initial guess, and the remaining $\{f_m(x)\}_{m=1}^{M}$ are incremental functions – steps or boosts – defined by the optimization method.

To use gradient descent on the empirical risk, we need to compute its negative gradient, which we denote $\mathbf{u}$. There are two ways to arrive at that. One is to look calculate partial derivatives of the empirical risk (3.1), with

respect to each estimated function value $\hat{f}_{m-1}(x_i)$:

$$\mathbf{u} = -\left(\frac{\partial}{\partial f(x_1)} R(f(x_1)), \ldots, \frac{\partial}{\partial f(x_n)} R(f(x_n))\right) \tag{3.6}$$

$$= -\left(\frac{\partial}{\partial f(x_1)} \sum_{i=1}^{N} \rho(y_i, f(x_i)), \ldots, \frac{\partial}{\partial f(x_n)} \sum_{i=1}^{N} \rho(y_i, f(x_i))\right) \tag{3.7}$$

$$= -\left(\frac{\partial}{\partial f(x_1)} \rho(y_1, f(x_1)), \ldots, \frac{\partial}{\partial f(x_n)} \rho(y_n, f(x_n))\right) \tag{3.8}$$

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

$$\mathbf{u} = \left(-\frac{\partial}{\partial \hat{f}(x)} \rho(y_i, \hat{f}(x))\big|_{\hat{f}=\hat{f}_{m-1}}\right)_{i=1}^{N} \tag{3.9}$$

This $\mathbf{u}$ is a vector of *generalized residuals*. We could also have arrived at it simply by 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 work, 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. And as statisticians we in any case wish to have an interpretable model. So we choose a model class

$$h(\cdot; \boldsymbol{\beta}) \tag{3.10}$$

which is parameterized by $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_p)$. If h is relatively simple, we call it a base learner. A good example of a base learner is a linear least squares model. These base learners are usually relatively simple, regularized (see definition of regularization) parametric effects of β_j. Typical examples are such as linear least squares, stumps (trees with one split; see Bühlmann and Hothorn (2007) and Hastie et al. (2009)), or splines with a few degrees of freedom. When using an additive model, it is reasonable to let the individual h_m's be simple models, because often algorithms exist for very fast computation of estimates. It is not as easy to get a gain through combining many complex learners.

We as mentioned start with an initial guess $f_0(\cdot)$, a constant, say. Then we iterate, at each step first calculating the generalized residuals from the previous iteration. At a given step in our gradient descent on the empirical risk, we as mentioned wish to minimize the generalized residuals. We constrain our choice of functions to the parameterized function class $h(x; \boldsymbol{\beta})$. The function that we wish to choose, which minimizes the residuals, is the member of that parameterized class that produces the $\hat{h}$ which is *most parallel* to u. This is the h that is most correlated with $\mathbf{u}$ over the data distribution. This means that this $\hat{h}_m$ is an approximation of the generalized residuals $\mathbf{u}$, or, a projection of the generalized residuals onto the space spanned by the base learner function class. We obtain that $\hat{h}_m$ by solving

$$\boldsymbol{\beta}_m = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \sum_{i=1}^{N} (u_i - h(x_i; \boldsymbol{\beta}))^2, \tag{3.11}$$

i.e., choose the h which minimizes the residual sum of squares (RSS) on the generalized residuals. We obtain $\hat{h}_m(\cdot; \boldsymbol{\beta}_m)$, the function to add into our model. The current model is $\hat{f}_{m-1}$. We do a line search to find which the best step length to use,

$$a_m = \underset{a}{\operatorname{argmin}}\, R(\hat{f}_{m-1} + a \cdot \hat{h}_m \cdot; \boldsymbol{\beta}).$$

The final model is then the sum of these terms. We can use gradient descent to find the parameters in each iteration, i.e., to find each h_m. In other words, doing gradient descent in parameter space (Friedman, 2001). So boosting can be viewed as an optimization procedure in functional space. This concludes the outline of a generic functional gradient descent algorithm. For a schematic overview, see Algorithm 2.

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 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 $\text{sl} = 0.1$ (Mayr et al., 2014). 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.

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. We denote the resulting parameter m_{stop}. If m_{stop} is too small, the model will underfit and it cannot fully incorporate the influence of the effects on the response and will consequently have poor performance. On the other hand, too many iterations will result in overfitting, leading to poor generalization.

Loss functions

Gradient boosting requires only that one uses a convex and differentiable loss function. The loss function measures the difference, or distance, between the true outcome y and the predicted outcome. A loss function must be symmetric, since it doesn't matter which is the prediction and which is the truth, and it must be convex, since the loss for a prediction which is further away from

algo:fgd

Algorithm 2 Functional gradient descent

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,

$$\hat{f} = \underset{f}{\operatorname{argmin}} R(f) = \underset{f}{\operatorname{argmin}} \sum_{i=1}^{n} \rho(y^{(i)}, f(x^{(i)})). \tag{3.12}$$

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.,

$$f_0(\cdot) = \underset{c}{\operatorname{argmin}} R(c). \tag{3.13}$$

3. Specify a base learner class h.

4. Increase m by 1.

5. Compute the negative gradient vector,

$$\mathbf{u} = \left(-\frac{\partial}{\partial \hat{f}} \rho(y_i, \hat{f}(x))\big|_{\hat{f}=\hat{f}_{m-1}} \right)_{i=1}^{N} \tag{3.14}$$

6. Estimate $\hat{h}_m$ by fitting $(\mathbf{X}_i, U_i)$ using the base learner h (like in the previous algorithm):

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

   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. Find best step length a_m by a line search:

$$a_m = \underset{a}{\operatorname{argmin}} R(\hat{f}_{m-1} + a \cdot \hat{h}_m \cdot; \boldsymbol{\beta}).$$

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

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

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

another point at least cannot be smaller. Examples of choices for ρ are the absolute loss $|y - \eta(x)|$, which leads to a regression model for the median, and the quadratic loss (the L_2 loss), which leads to the usual regression model for the mean. Very often, the loss is derived from the **negative** log likelihood of the distribution of $\mathcal{Y}$, depending on the desired model. 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. In the survival data setting, typical loss functions are ROC and Briar score (Bøvelstad and Borgan, 2011).

Practical considerations

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

3.4 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 by (Tukey, 1977), who called it "twicing". 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!!

L_2Boost example

Lorem ipsum.

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,

$$\hat{f} = \underset{f}{\operatorname{argmin}} R(f) = \underset{f}{\operatorname{argmin}} \sum_{i=1}^{n} y_i - \hat{f}(x_i). \tag{3.15}$$

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.,

$$f_0(\cdot) = \underset{c}{\operatorname{argmin}} R(c). \tag{3.16}$$

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

$$h(\mathbf{x}, \boldsymbol{\beta}) = \mathbf{x}^T \boldsymbol{\beta} = \sum_{j=1}^{p} \beta_j x_j \tag{3.17}$$

4. Increase m by 1.

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

$$\mathbf{u} = \left(y_i - \hat{f}(x_i)\right)_{i=1}^{N} \tag{3.18}$$

6. Estimate $\hat{h}_m$ by fitting $(\mathbf{X}_i, U_i)$ using the base learner h (like in the previous algorithm):

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

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

3.5 High dimensions and component-wise gradient boosting

In some situations, a data set consists of more predictors p than observations N. We often call this the $p > N$ setting, or simply a high-dimensional setting. With such data sets, it will be infeasible to use a base learner h which incorporates all p dimensions of the observation matrix $\mathbf{X}$. 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.

Component-wise gradient boosting is an algorithm which does work 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 L2boost paper by Bühlmann and Yu (2003), and it has further been refined and explored, see e.g. Bühlmann (2006). In a gradient boosting algorithm, we start out with an empty model $f_0(\cdot)$, which perhaps only uses a constant, or something of the sort. We have not added any predictors yet. Instead of adding a small effect from all predictors, as above, we could try adding only one variable at a time. This is a typical statistical model selection regime, namely forward stepwise model selection. Here one at each step looks at all predictors separately, and tries adding it to the model. One proceeds only with the predictor which gives the best improvement.

The main idea of component-wise gradient boosting is to do exactly this, except in a stagewise manner, for boosting. While stepwise procedures look at all parameters in a model at the same time, a stagewise does not change the added parameters, but only looks at the next one. The component-wise gradient boosting algorithm takes the generic functional gradient boosting algorithm (2) and 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 the structure of these base learner is exactly the same, but there is one base learner for each predictor. So if using a linear least squares model, the set of base learners would be

$$\mathcal{H} = \{h_1(\mathbf{x}; \beta) = \beta x_1, h_2(\mathbf{x}; \beta) = \beta x_2, \ldots, h_p(\mathbf{x}; \beta) = \beta x_p\}. \tag{3.19}$$

One then fits all of these separately to the generalized residuals $\mathbf{u}$, and selects only one learner, that with the best performance, to be added into the boosted model. The resulting model

$$f(\cdot) = \sum_{m=1}^{M} f_m(\cdot), \tag{3.20}$$

where each f_m is a componentwise learner, can then also be seen as a sum of componentwise effects,

$$f(\mathbf{x}) = \sum_{j=1}^{p} p_j(x_j),$$

where $p_j(\cdot) = \sum_{m=1}^{M} f(x_j)$. For a schematic overview of the algorithm, see Algorithm 4.

algo:component-gradboost

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 $\rho(y, \hat{f}(x))$, for which we wish to minimize the empirical risk, i.e., the loss function evaluated on the samples,
$$\hat{f} = \underset{f}{\operatorname{argmin}} R(f). \tag{3.21}$$
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.,
$$f_0(\cdot) = \underset{c}{\operatorname{argmin}} R(c). \tag{3.22}$$
3. Specify a set of base learners $\mathcal{H} = \{h_1(\cdot), \ldots, h_p(\cdot)\}$, where each h_j is univariate.
4. Increase m by 1.
5. Compute the negative gradient vector,
$$\mathbf{u} = \left(-\frac{\partial}{\partial \hat{f}} \rho(y_i, \hat{f}(x)) \big|_{\hat{f}=\hat{f}_{m-1}} \right)_{i=1}^{N} \tag{3.23}$$
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
$$\boldsymbol{\beta}_m = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \sum_{i=1}^{N} (u_i, h_j(\mathbf{x}_i; \boldsymbol{\beta}))^2$$
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. Select $h(\cdot; \boldsymbol{\beta}_m) = \operatorname{argmin}_j \sum_{i=1}^{N} (u_i, h_j(\mathbf{x}_i; \boldsymbol{\beta}))^2$
8. Update $f_m(\cdot) = f_{m-1}(\cdot) + h(\cdot; \boldsymbol{\beta}_m)$.
9. Repeat steps 4 to 8 (inclusive) until $m = M$.
10. Return $\hat{f}(\cdot) = \hat{f}_M(\cdot) = \sum_{m=0}^{M} f_m(\cdot)$.

Variable selection

If the number of iterations M 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, many of the base learners 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. In many cases, a good M is less than the number of predictors, and so even if every iteration selects a different base learner, the final model will have excluded predictors. Realistically, some predictors will be more explanatory than others, and so they will be selected more than once. Some predictors are simply 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.

3.6 Selecting $m_{\mathbf{stop}}$

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. Left on its own, the parameters in boosting will converge towards the maximum likelihood parameters, 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 etc. We argue instead that cross-validation, a very common method for selection of tuning parameters in statistics, is a good method to use. It is flexible and easy to implement. It is somewhat computationally demanding, requiring several full runs of the boosting algorithm.

SOURCE?

Other selection methods

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

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 fold k. This procedure is repeated for all

$k = 1, \ldots, K$. An estimate for the prediction error is obtained by summing over the test errors from evaluting the left-out fold. Let $\kappa(k)$ be the set of indices for fold k. The cross-validated estimate for a given m then becomes

$$\text{CV}(m) = \sum_{k=1}^{K} \sum_{i \in \kappa(k)} \text{L}\left((t_i, \delta_i), \theta_m; \mathbf{x}_i\right). \tag{3.24}$$

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,

$$m_{\text{stop}} = \underset{m}{\text{argmin}}\, \text{CV}(m). \tag{3.25}$$

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 one evaluates the model on the observation that was left out. In this case, the outcome is deterministic, since there is no randomness when dividing into folds.

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 "real-life data sets like ours", Kohavi (1995) illustrate that 10-fold stratified cross validation performs best.

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., 2016). Seibold et al. (2016) 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,

$$\text{RCV}(m) = \sum_{j=1}^{J} \sum_{k=1}^{K} \sum_{i \in \kappa(j,k)} \text{L}\left((t_i, \delta_i), \theta_m; \mathbf{x}_i\right). \tag{3.26}$$

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

$$m_{\text{stop}} = \underset{m}{\text{argmin}}\, \text{RCV}(m). \tag{3.27}$$

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.

3.7 Multidimensional boosting: (Component-wise) boosting of a multivariate loss function

The above methods consider a loss function depending on one parameter: $\mathrm{L}(\beta)$. In the boosting steps, one uses a gradient descent step with the loss function differentiated with respect to this one parameter. But this means that we are restricted to boosting models which have only one parameter. In many applications, this will not be sufficient, and will not be flexible enough. We want to be able to boost models of more than one parameter, i.e., multidimensional boosting.

Cyclical multidimensional boosting

Schmid et al. (2010) extend the component-wise gradient boosting algorithm (Friedman, 2001) to such a setting where one has a multivariate loss function, a loss function with K parameters. Like before, we take the derivative of the loss function to get our generalized residuals. This time, we take the partial derivative of the multivariate loss function with respect to each variable. This results in K sets of generalized residuals, one for each input dimension. The algorithm proposed in Schmid et al. (2010) is to do boost each variable in each boosting step, i.e., in essence replicates the gradient boosting. See ... for a schematic overview. Note that this algorithm resembles the backfitting strategy by Hastie, Tibshirani (1990). 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. After having obtained an update of the parameters, the algorithm cycles through the scale/nuisance parameters, and maximizes these numerically.

The main tuning parameters here 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. Similarly, the step length should be small, but is not of minor importance.

In a boosting algorithm developed for the GAMLSS family (Rigby and Stasinopoulos, 2005), Mayr et al. (2012a) applies the above algorithm, with nuisance/scale parameter set equal to the empty set, i.e., they drop the entire steps X until X.

Noncyclical boosting algorithm

Recently, Thomas et al. (2018) point out a problem with the above approach: The different $m_{\text{stop},j}$ parameters are not independent of each other, and hence have to be jointly optimized. The usually applied *grid search* for such parameters scales exponentially with the number of parameters k, and this can quickly become computationally very demanding. Therefore Thomas et al. (2018)

algo:multi-cyclical

Algorithm 5 Multidimensional cyclical component-wise gradient boosting

1. Start with a data set $D = \{x_i, y_i\}_{i=1}^N$ and a chosen loss function $\rho(y, \hat{f}(x))$, for which we wish to minimize the empirical risk, i.e., the loss function evaluated on the samples,
$$\hat{f} = \underset{f}{\operatorname{argmin}} R(f). \tag{3.28}$$
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. Initialize scale (nuisance) parametes σ.
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. If $m > m_{\text{stop},k}$, go to step X. Otherwise 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 the negative gradient vector
$$\mathbf{u}_k^{(m-1)} = \left(-\frac{\partial}{\partial \hat{f}_k} \rho(y_i, \hat{f}^{(m-1)}(x_i)) \right)_{i=1}^N \tag{3.29}$$
7. 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 $\mathbf{u}_k^{(m-1)}$.
8. Select the component of X which best fits $\mathbf{u}_k^{(m-1)}$ according to a pre-specified goodness-of-fit, typically RSS. Set $\hat{\mathbf{u}}_k^{(m-1)}$ equal to the fitted values of the corresponding best model fitted in step X.
9. Update $\hat{f}_k[m-1] \leftarrow \hat{f}_k^{[m-1]} + \nu \hat{U}_k^{[m-1]}$, where ν is a pre-specified real-valued step-length factor.
10. Repeat steps X until X for $k = 2, \ldots, K$. Update
11. Repeat steps 4 to 8 (inclusive) until $m = M$.
12. Finally, update $\hat{f}^{[m]} \leftarrow \hat{f}^{[m-1]}$.
13. Set $l = 0$.
14. Increase l by 1.
15. 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.
16. Repeat steps X until X for $l = 2, \ldots, L$.
17. Repeat steps ... until $m = max_k(m_{\text{stop},k})$.
18. Return $\hat{f}(\cdot) = \hat{f}_M(\cdot) = \sum_{m=0}^M f_m(\cdot)$.

develop an alternative solution in which only one tuning parameter is needed. They call the approach in the above algorithm for *cyclical* multidimensional boosting (one cycles through all parameters in each iteration). Thomas et al. (2018) develop an algorithm where instead of cyclical boosting, one chooses only one parameter in each boosting iteration.

Usually, base-learners are selected by comparing their residual sum of squares with respect to the negative gradient vector. Thomas et al. (2018) call this the *inner loss*. However, in general these are not comparable across parameters of the loss function.Because this depends on the scale of the parameters. In a multivariate normal distribution, for example, the partial derivatives for each mean are symmetrical. But the partial derivative for the standard deviation will not be comparable. Therefore, to compare between parameters, a different comparison method is needed. Thomas et al. (2018) give two methods here, which they call *inner* and *outer* loss. 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.

Add small section with empirical proof?

Inner loss

One solution is to compare the empirical risks after the update with the best-fitting base-learners that have been selected via the residual sum of squares for each distribution parameter. In other words, for each parameter, calculate the estimated base learner for each base learner. Then, choose the base learner which has the best RSS of the generalized residuals. Finally, calculate the empirical risk for the model, given that you choose each of these base learners separately. Compare the additional gain $\Delta\rho$ across the selected base learners. Then, select only that parameter and that base learner which led to the best improvement, and update the model accordingly.

Outer loss

Another option is to use the empirical risk for choosing the base learner within the parameter as well. Choosing base-learners and parameters with respect to two different optimization criteria may not always lead to the best possible update. The empirical loss, i.e., the negative log likelihood of the modeled distribution can be used to compare both. The negative gradients are used to estimate all base-learners. The improvement in the empirical risk 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.

Here I could perhaps write something about the performance of the noncyclical compared with the cyclical.

Degenerate noncyclical boosting algorithm

We have a loss function for which we have several partial derivatives, one for each component/parameter. One way to extend the component-wise gradient boosting algorithm for one-dimensional loss functions into several dimensions is

to estimate all base learners for all loss function components, and simply choose the best component according to the same criterion as for one-dimensional loss function, namely that with best RSS. This is however not a good idea, because the gradients are **not comparable** (Thomas et al., 2018). Therefore, Thomas et al. (2018) give the following algorithm.

algo:multi-non-cyclical

Algorithm 6 Multidimensional non-cyclical component-wise gradient boosting

1. Start with a data set $D = \{x_i, y_i\}_{i=1}^N$ and a chosen loss function $\rho(y, \hat{f}(x))$, for which we wish to minimize the empirical risk, i.e., the loss function evaluated on the samples,
$$\hat{f} = \underset{f}{\operatorname{argmin}} R(f). \tag{3.30}$$
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
$$\mathbf{u}_k^{(m-1)} = \left(-\frac{\partial}{\partial \hat{f}_k} \rho(y_i, \hat{f}^{(m-1)}(x_i)) \right)_{i=1}^N \tag{3.31}$$
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
   $$j^* = \underset{j \in 1, \ldots, J_k}{\operatorname{argmin}} \sum_{i=1}^N (u_k^{(i)} - \hat{h}_{kj}(x^{(i)}))^2 \tag{3.32}$$
   - the outer loss, i.e., the loss function after the potential update,
   $$j^* = \underset{j \in 1, \ldots, J_k}{\operatorname{argmin}} \sum_{i=1}^N \rho \left(y^{(i)}, \hat{f}^{(m-1)}(x^{(i)}) + \nu \cdot \hat{h}_{kj}(x^{(i)}) \right) \tag{3.33}$$
10. Compute the possible improvement of this update regarding the outer loss,
$$\Delta\rho_k = \sum_{i=1}^N \rho \left(y^{(i)}, \hat{f}^{(m-1)}(x^{(i)}) + \nu \cdot \hat{h}_{kj^*}(x^{(i)}) \right) \tag{3.34}$$
11. Update, depending on the value of the loss reduction, $k^* = \operatorname{argmin}_{k \in 1, \ldots, K} \Delta\rho_k$
$$\hat{f}_{k^*}^{(m)} = \hat{f}_{k^*}^{(m-1)} + \nu \cdot \hat{h}_{k^* j^*}(x), \tag{3.35}$$
while for all $k \neq k^*$,
$$\hat{f}_{k^*}^{(m)} = \hat{f}_{k^*}^{(m-1)}. \tag{3.36}$$
12. Repeat steps ... 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. This does not pose a problem. Summary of the procedure:

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.

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.

algo:FHT-sim

Algorithm 7 Generating survival data from Inverse Gaussian FHT distribution

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

Derivatives not on same scale

Pass.

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.

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 7, with the censoring time W being drawn from a distribution $\exp(0.1)$. The resulting survival times have the following Kaplan-Meier plot.

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 $U_{\mu}^{[m-1]}$.

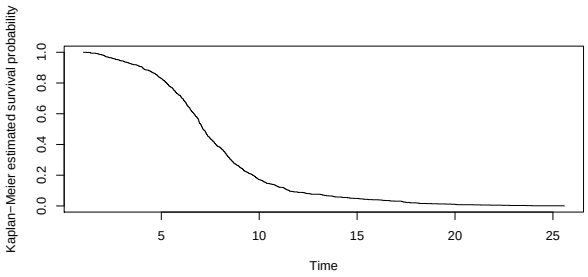
   g) Select the component of $\mathbf{Z}$ which best fits $U_{\mu}[m-1]$ according to R^2. Set $\hat{U}_{\mu}^{[m-1]}$ equal to the fitted values of the corresponding best model fitted in the previous step.

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

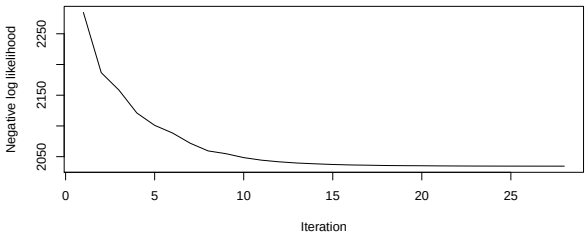
   i) Update $\hat{f}^{[m]} \leftarrow \hat{f}^{[m-1]}$.

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

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



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.



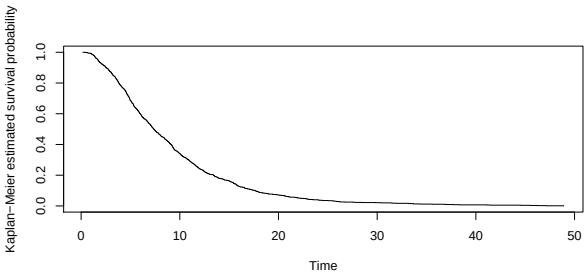
The final $\hat{\boldsymbol{\beta}}$ is $(1.968, 0.103, 0.180)$, and the final $\hat{\boldsymbol{\gamma}}$ is $(-0.964, -0.082, 0.062)$. The parameters found by numerically maximizing the joint maximum likelihood are also included. Summarized in the table below.

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.

Large example with uncorrelated matrices

Here, N is 1000. We let $\boldsymbol{\beta}$ be a large vector of size $p = 10001$, and $\boldsymbol{\gamma}$ be a small vector of size $d = 16$. We imagine X, corresponding to $\boldsymbol{\beta}$, be gene expressions, whereas Z, corresponding to $\boldsymbol{\gamma}$ be clinical measurements. 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. We also here draw X and Z from a beta distribution. 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 80. We then run our algorithm with that number of iterations. We find that elements ... are selected. They are

Large example with correlated matrices

Here, N is 1000. We let $\boldsymbol{\beta}$ be a large vector of size $p = 10001$, and $\boldsymbol{\gamma}$ be a small vector of size $d = 16$. We imagine X, corresponding to $\boldsymbol{\beta}$, be gene expressions, whereas Z, corresponding to $\boldsymbol{\gamma}$ be clinical measurements. 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.

Chapter 5

Model fitting

When looking at a specific data set of individuals with covariates and corresponding responses, a statistician wishes to understand their relationship. In other words, she assumes that there actually is some structured relationship between the input and the output. This relationship is something we wish to model. A model is a simplification or approximation of reality and hence will not reflect all of reality. As the statistician George Box's aphorism goes: All models are wrong, but some are useful. We assume the data set consists of n individuals, where each individual $\mathbf{x}_i$, $i = 1, \ldots, n$, has p covariates: $x_{i1}, x_{i2}, \ldots, x_{ip}$. We express these in vector form

$$\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip}) .$$

The individual has a corresponding real-valued output y_i. The statistician will assume that the relationship between an covariate vector x_i and an output value y_i can be expressed as

$$y_i = f(\mathbf{x}_i) + \varepsilon,$$

where $f(\cdot)$ is some model which the statistician will have to choose, and ε is the error term, the difference between the value which the statistician's model gives, and the actual output. This model is perfectly general. Choosing a model for $f(\cdot)$ is what most of statistics is about. In the machine learning field, one has typically been more focused on choosing a model which has a low error term. But one has not necessarily cared about the model $f(\cdot)$ being interpretable. Being interpretable means that you can say something about the effect of an increase in a specific covariate, say. We call these non-interpretable models for black-box models. To be sure, black box models will often perform incredibly well in predicting an outcome which is correct. In statistics, however, we are more interested in a model which is interpretable, such that we can quantify the relation between one or more predictor variables and the expectation of the response.

5.1 Model selection

Given that all models are wrong, it is easy to see that there is no perfect model for a data set. But some are better than others. There exist many selection

criteria one can use to do this, such as the Akaike Information Criterion (AIC) and the Bayes Information Criterion (BIC). In general, though, what one wishes to do in all cases is have the model result in as low error terms as possible, while at the same time not overfitting. One way to think of overfitting is that the model starts to learn the structure of the error terms. One might think of the manydimensional space of the input matrix $\mathbf{X}$, which has the individuals $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ as rows, where the corresponding outputs of the individuals are points in this space. If a model is overfitting, the curvature of the model in this space is very high. If the model has no curvature, it means that the output will be the same for all individuals, so this will also entail a very high error. In other words, this will not be a good model. In general, one wishes to have low error terms, and low curvature. This is a trade-off one has to make which is called the bias-variance tradeoff. In the case of overfitting, where one has high curvature, there is a high variance between predictions. Even if one takes one observation and shifts it somewhat, then the predicted value will be very different. On the other hand, a flat curvature model has no variance, but very high bias. One can understand, then, that there exists a middle ground which is a good compromise between the two: A model which has both reasonably low bias and reasonably low variance.

5.2 Variable selection

Having chosen a specific type of model – a linear regression, say – an important next step is to select variables to use in our model $f(\cdot)$. This can be done in many ways. One way is to try all possible combinations. The number of combinations explodes combinatorially with the number of predictors, since the number of possible combinations is the factorial $p!$ of p. Two other main methods are often used in the case of reasonably few predictors. One is to start with the null model, that is, no predictors. Then, for all predictors, one can try adding it to the model. One proceeds with the predictor which gives the best improvement. This is called forward stepwise variable selection. What easily comes to mind now is to do the reverse, namely backwards stepwise variable selection. Start with a model which contains all predictors. In each step, remove that predictor which gives the least improvement compared to the model without that predictor. Ideally, one chooses a stopping criterion. In both cases, one typically chooses a significance level $1 - \alpha$ which is a stopping point for the procedure. This means that in the forward stepwise method, one stops when there are no predictors with a p-value of less than α left to add into the model. In the case of backwards stepwise, then, there are no predictors with a p-value of more than α left. One could also, of course, use other types of criteria for stopping the procedure, for instance a regular model selection criterion such as the aforementioned AIC or BIC. Note that in each step, with a given set of predictors, one fits the entire model anew, meaning parameter values for a given predictor will be different in each step. This is different from a stagewise approach which is what boosting uses.

With a model $f(\cdot)$, we have something which can predict the response variable y for any given $\mathbf{x}$. We can call this prediction for $\hat{y}$. The error term ε is then $y - \hat{y}$. To calculate the difference between the estimated outcome and the actual outcome, we need a loss function ρ. It measures the difference,

or distance, between the true outcome y and the predicted ou. A proper loss function must be symmetric and convex. Examples of choices for ρ are the absolute loss $|y - \eta(x)|$, which leads to a regression model for the median, and the quadratic loss (the L_2 loss), which leads to the usual regression model for the mean. Very often, the loss is derived from the **negative** log likelihood of the distribution of $\mathcal{Y}$, depending on the desired model. 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. In the survival data setting, typical loss functions are ROC and Briar score (Bøvelstad and Borgan, 2011).

5.3 Combatting overfitting

5.4 High-dimensional data

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

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

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

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.

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.

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.

hastie2007
Hastie, T. (2007). Comment: Boosting algorithms: Regularization, prediction and model fitting. *Statist. Sci.*, 22(4):513–515.

ESL
Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer.

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.

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.

mayr14a
Mayr, A., Binder, H., Gefeller, O., and Schmid, M. (2014). The evolution of boosting algorithms. from machine learning to statistical modelling. *Methods of Information in Medicine*, 53(6):419–427.

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.

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 Bin, R. D. (2016). On the choice and influence of the number of boosting steps.

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.

tukey

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

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.

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.