Boosting the First-Hitting-Time Regression Model

Vegard Stikbakke

April 25, 2018

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

sec:intro

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

CHAPTER 2

First hitting time regression models

2.1 Survival analysis and time-to-event models

sec:survival

In many fields, it is interesting to consider the lifetime of some entity. A lifetime ends when an event occurs. We are usually interested in inferring things about this lifetime, and what it depends upon. In medical fields, this is called survival analysis, while in engineering it is called reliability analysis. In the former case, we consider the lifetime of patients or the length of a hospital stay after some treatment. In the latter, we consider, e.g., the time before a component of a system breaks and must be replaced.

The time-to-event T is a continuous, non-negative random variable $T \sim f(t)$, $t > 0$, for some probability density function f. We are particularly interested in two things related to T:

1. The survival function $S(t)$ – the probability of an individual having survived until time t. Note that $S(t) = 1 - F(t)$, where F is the cumulative density function of f.

2. The hazard function $h(t)$ – the probability of the event happening at time t. Note that this is conditional on surviving until time t, and is defined as $h(t) = \frac{f(t)}{S(t)}$.

Regression

sec:surv-reg

To find out anything interesting, we need to be able to do regression on covariates. Given a sample of n independent observations $\{t_i, \mathbf{x}_i, \delta_i, i = 1, \ldots, n\}$, where individual i has covariates $\mathbf{x}_i$, lifetime t_i and censoring indicator $\delta_i, i = 1, \ldots, n$, which is 1 if the event has happened, and 0 if not. From Caroni 2017, p. 10, the likelihood is given by

$$L(\boldsymbol{\theta}|\mathbf{x}_1, \ldots, \mathbf{x}_n) = \prod_{i=1}^{n} f(t_i|\mathbf{x}_i, \boldsymbol{\theta})^{\delta_i} S(t_i|\mathbf{x}_i, \boldsymbol{\theta})^{1-\delta_i} \tag{2.1}$$

{eq:surv-lik}

Proportional hazards

The most used method for doing regression on survival data is the Cox proportional hazards (PH) regression. It is based on an assumption that is often called the PH property or the PH assumption, namely that

$$h(t|x) = h_0(t)g(\mathbf{x}), \tag{2.2}$$

where $h_0(t)$ is a baseline hazard function.

more about baseline hazard?

This property states that at any two time points t_1 and t_2, the ratio between the hazard functions of any two $\mathbf{x}_1$ and $\mathbf{x}_2$ will be the same:

$$\frac{h(t_1|x_1)}{h(t_1|x_2)} = \frac{h(t_2|x_1)}{h(t_2|x_2)} \tag{2.3}$$

This is a strong assumption to make, and it will rarely be the case in practice (Lee and Whitmore 2010). However, many times Cox regression will work well in practice.

how to rephrase?

really? or argue why!

2.2 The first hitting time model

sec:fht

Revisiting the examples of the two lifetime settings, it may in both cases be natural to imagine that the event happens as a process reaches a threshold. Then one way to model the time-to-event is to model the process itself, and look at the time it takes for the process to reach this threshold, at which point the event is triggered. Lee and Whitmore 2006 is a thorough review on the first hitting time model, and Caroni 2017 is a book which covers many aspects of it. We continue by describing the first hitting time model.

An first hitting time model has two main components.

1. A stochastic process $\{Y(t), t \in \mathcal{T}, y \in \mathcal{Y}\}$, with $Y(0) = y_0$.
2. A boundary set, $B \subset \mathcal{Y}$, where $y_0 \notin \mathcal{B}$

The first hitting time is the first time the process reaches the boundary set. Formally, the first hitting time is a stochastic variable S, which is defined as

what did you say about this again?

$$S = \inf\{t \colon Y(t) \in \mathcal{B}\}$$

Typically, one will consider a process with boundary $B = 0$. The event then occurs if and when the process $\{Y\}$ reaches 0 at $y(S)$. Note that it is possible that $P(S < \infty) < 1$.

justify

The first hitting time model is conceptually appealing and does not require the PH assumption, and is hence more flexible. In fact, the PH model may be obtained by constructing the first hitting time model in a specific way (Lee and Whitmore 2010).

Different choices of processes lead to different kinds of distributions for the first hitting time. We now look at a common choice of the process.

Wiener process

sec:wiener

The Wiener process, also known as the standard Brownian motion process, is a process which is continuous in time and space, and has the properties (Caroni 2017, p. 61) that

- $Y(t)$ has independent increments, such that $Y(t_2)-Y(t_1)$ and $Y(t_4)-Y(t_3)$ are independent for any disjoint intervals, and
- for any interval (t_1, t_2),

$$Y(t_2) - Y(t_1) \sim N(\mu(t_2 - t_1), \sigma^2(t_2 - t_1)).$$

This is a process which will both increase and decrease. However, if we want a monotonic restriction on the movement of the process, we may use a gamma process.

Gamma process

The gamma process is suitable for modelling a process which we would require to be monotonic, typically a physical degradation, i.e. where the damage cannot mend itself, unlike a patient's health. The first-hitting-time that arises from the gamma process is inverse gamma. (Lee and Whitmore 2006, p. 503.)

make this into a separate section?

Other choices of processes include Markov chain state models, the Bernoulli process, and the Ornstein-Uhlenbeck process.

2.3 First hitting time regression based on underlying Wiener process

The first hitting time of the Wiener process (section 2.2) follows an inverse Gaussian distribution (derivation in Chhikara 1988, pp. 23-29):

also derive more clearly in appendix?

$$f(t|y_0, \mu, \sigma^2) = \frac{y_0}{\sqrt{2\pi\sigma^2 t^3}} \exp\left[-\frac{(y_0 + \mu t)^2}{2\sigma^2 t}\right] \tag{2.4}$$

{eq:fht-ig}

If μ is positive, $Y(t) \leq 0$ is not certain to occur. Note also that this model is over-parameterized, because Y has an arbitrary scale, so we can without loss of generality set $\sigma^2 = 1$.

more!

While μ and y_0 have simple interpretations in terms of the underlying process, they do not in terms of the lifetime distribution. The mean lifetime is $\frac{y_0}{|\mu|}$, and the variance is $\frac{y_0}{|\mu|^3}$. (Caroni 2017, p. 62.)

The cumulative distribution function of the FHT is (from Xiao et al. 2015, p. 7)

$$F(t|\mu, \sigma^2, y_0) = \Phi\left[(-\frac{y_0 + \mu t)}{\sqrt{\sigma^2 t}}\right] + \exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left[\frac{\mu t - y_0}{\sqrt{\sigma^2 t}}\right], \tag{2.5}$$

{eq:cumulative}

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

$$\Phi(x) = \int_{-\infty}^{x} \exp(-y^2/2)/\sqrt{2\pi}\, \mathrm{d}y. \tag{2.6}$$

Regression

We may introduce effects from covariates by allowing μ and y_0 to depend on covariates $\mathbf{x}$. Suitable models are

$$\begin{aligned} \mu &= \boldsymbol{\beta}^{\mathrm{T}}\mathbf{x} \\ \ln y_0 &= \boldsymbol{\gamma}^{\mathrm{T}}\mathbf{z} \end{aligned} \tag{2.7}$$

{eq:coeffs}

where $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ are vectors of regression coefficients. Note that we may let $\mathbf{x}$ and $\mathbf{z}$ share none, some, or all elements.

2.4 Likelihood

sec:lik

In section 2.1, we stated the likelihood of lifetime regression models in (2.1). For an inverse gaussian FHT this then becomes (inserting (2.4) and (2.5) into (2.1), and since $S(t) = 1 - F(t)$)

$$\begin{aligned} L(\boldsymbol{\theta}|\mathbf{x}_1,\ldots,\mathbf{x}_n) = &\left(\frac{y_0}{\sqrt{2\pi\sigma^2 t^3}}\exp\left[-\frac{(y_0+\mu t)^2}{2\sigma^2 t}\right]\right)^{\delta_i} \\ &\times\left[1-\Phi\left(-\frac{y_0+\mu t}{\sqrt{\sigma^2 t}}\right)-\exp\left(-\frac{2y_0\mu}{\sigma^2}\right)\Phi\left(\frac{\mu t - y_0}{\sqrt{\sigma^2 t}}\right)\right]^{1-\delta_i} \end{aligned} \tag{2.8}$$

{eq:fht-lik}

Since we let $\sigma^2 = 1$, this simplifies to

$$\begin{aligned} L(\boldsymbol{\theta}|\mathbf{x}_1,\ldots,\mathbf{x}_n) = &\left(\frac{y_0}{\sqrt{2\pi t^3}}\exp\left[-\frac{(y_0+\mu t)^2}{2t}\right]\right)^{\delta_i} \\ &\times\left[1-\Phi\left(-\frac{y_0+\mu t}{\sqrt{t}}\right)-\exp(-2y_0\mu)\Phi\left(\frac{\mu t - y_0}{\sqrt{t}}\right)\right]^{1-\delta_i} \end{aligned} \tag{2.9}$$

We can now substitute the covariates in (2.7) into this. To find optimal parameters, we use numerical maximum likelihood methods. However, this is only feasible in the low-dimensional case, since it will optimize the entire parameter space at once. Therefore it is necessary to develop methods which can deal with high-dimensional cases. That is what we intend to do in the main part of the thesis.

is this correct?

2.5 The `threg` package

There exists an R package `threg` for fitting regression with inverse gaussian FHT, described in Xiao et al. 2015. We provide a small example here, which is the one described in the help pages of the package.

```r
library(threg)
data("lkr")
lkr$f.treatment2=factor(lkr$treatment2)
# head(lkr)
fit <- threg(Surv(weeks, relapse) ~ f.treatment2|f.treatment2,
    data=lkr)
fit
```

Which provides the following output

```
Call:
threg(formula = Surv(weeks, relapse) ~ f.treatment2 | f.treatment2,
    data = lkr)

                              coef   se(coef)         z       p
lny0: (Intercept)     2.0097844 0.1705141 11.786620 0.0e+00
lny0: f.treatment21 -1.2739233 0.2441633 -5.217504 1.8e-07
  mu: (Intercept)   -0.5886165 0.1340126 -4.392246 1.1e-05
  mu: f.treatment21  0.5888365 0.1535081  3.835866 1.3e-04

Log likelihood =-104.64, AIC =217.28
```

Here we fit an inverse gaussian FHT model where

$$\ln y_0 = \mu = \beta_0 + \beta_1 \mathrm{I}(\text{treatment2} = 1)$$

What the `threg` function in the package of the same name does, is essentially to set up the log likelihood and use the numerical optimization function `nlm` to find the optimal parameters.

Recreating

We recreate the above example in plain R code.

```
library(threg)
data("lkr")
lkr$f.treatment2=factor(lkr$treatment2)
fit <- threg(Surv(weeks, relapse) ~ f.treatment2|f.treatment2,
    data=lkr)

library(dplyr)
#tbl <- select(.data=lkr, weeks, relapse, f.treatment2)
tbl <- data.frame(lkr$weeks, lkr$relapse, lkr$f.treatment2,
                  row_names = c("weeks", "relapse", "f.treatment2"))
names(tbl)[names(tbl) == "lkr.weeks"] <- "weeks"
names(tbl)[names(tbl) == "lkr.relapse"] <- "relapse"
names(tbl)[names(tbl) == "lkr.f.treatment2"] <- "f.treatment2"

to_optimize <- function(params) {
  total_loglikelihood <- 0

  gamma <- params[1:2]
  beta <- params[3:4]

  for (i in 1:n) {
    tbl_i <- tbl[i, ]
    event <- tbl_i$relapse
    t_i <- tbl_i$weeks
    is_treated <- as.integer(tbl_i$f.treatment2)-1
    X_i <- c(1, is_treated)
    y0_i <- exp(sum(gamma*X_i))
    mu_i <- sum(beta*X_i)
    log_f_i <- log(y0_i) - 0.5*log(2*pi*t_i^3) - ((y0_i +
        mu_i*t_i)^2)/(2*t_i)
    log_S_i <- log(1 - pnorm(-(y0_i+mu_i*t_i)/sqrt(t_i)) -
        exp(-2*y0_i*mu_i)*pnorm((mu_i*t_i-y0_i)/sqrt(t_i)))
    loglik_i <- event*log_f_i + (1 - event)*log_S_i
    total_loglikelihood <- total_loglikelihood + loglik_i
  }
  return(-total_loglikelihood)
}

params_from_threg <- c(2.0098, -1.2739, -0.5886, 0.5886)
threg_value <- -to_optimize(params_from_threg)

initial_params <- c(1, 1, 1, 1)

best <- nlm(to_optimize, initial_params)
params_from_best <- best$estimate
best_value <- -best$minimum
print(best_value)
print(threg_value)
```

We can also inspect the parameters we found and see that we did indeed find the optimal parameters.

CHAPTER 3

Statistical learning theory

ch:learning-theory

Assume we have a joint distribution $(\mathbf{X}, Y)$, $X \in \mathbb{R}^p$ and $Y \in \mathbb{R}$, and $Y = F(\mathbf{X}) + \varepsilon$, $F(\mathbf{x}) = \mathrm{E}(Y|\mathbf{X} = \mathbf{x})$ We wish to estimate the underlying $F(\mathbf{X})$. For an estimate $\hat{F}(\cdot)$, we measure the loss, or the difference, with a loss function

$$\mathrm{L}(Y, \hat{F}(\mathbf{X})). \tag{3.1}$$

A common loss function for regression is the squared loss, also known as the L_2 norm,

$$\mathrm{L}(Y, \hat{F}(\mathbf{X})) = (Y - \hat{F}(\mathbf{X}))^2. \tag{3.2}$$

For a $\hat{F}(X)$, we wish to estimate the expected loss, also known as the generalization or test error,

$$\mathrm{Err}_\tau = \mathrm{E}[\mathrm{L}(Y, \hat{F}(\mathbf{X}))|\tau], \tag{3.3}$$

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

because?

$$\mathrm{Err} = \mathrm{E}[\mathrm{Err}_\tau] = \mathrm{E}_\tau\left([\mathrm{L}(Y, \hat{F}(\mathbf{X}))|\tau]\right), \tag{3.4}$$

{eq:err}

i.e., average over many different test sets. In practice, we observe a sample $(\mathbf{x}_i, y_i)_{i=1}^N$. For this sample, we can calculate the training error,

$$\overline{\mathrm{err}}(F) = \frac{1}{N}\sum_{i=1}^{N} \mathrm{L}(y_i, \hat{F}(\mathbf{x}_i)), \tag{3.5}$$

{eq:empirical-risk}

also known as the empirical risk. To estimate err (3.4), 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 train/fit/estimate based on the other 80%, and estimate Err by

$$\hat{\mathrm{Err}} = \frac{1}{M}\sum_{i=1}^{M} L(y_i, \hat{F}(\mathbf{x}_i)), \tag{3.6}$$

where (x_i, y_i) here are from the test set.

CHAPTER 4

Statistical boosting

The inception of boosting

Boosting is one of the most promising methodological approaches for data analysis developed in the last two decades. (Mayr et al. 2014) Boosting originated in the fields of computational learning theory and machine learning. In 1989 Kearns and Valiant, working on computational learning theory, posed a question: Could any weak learner be transformed to become also 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. Freund and Schapire invented the AdaBoost algorithm in 2006 for binary classification. (Freund and Schapire 1996) It was evidence that the answer to the original question was positive. The AdaBoost algorithm performs iterative reweighting of original observations. For each iteration, it gives more weight to misclassified observations, and then trains a new weak learner based on all weighted observations. It then adds the new weak learner to the final classifier. The resulting AdaBoost classifier is then a linear combination of these weak classifiers, i.e., a weighted majority vote. In its original formulation, the AdaBoost classifier does not have interpretable coefficients, and as such it is a so-called black-box algorithm. In statistics, however, we are interested in models which are interpretable.

4.1 Statistical boosting

sec:sboost

In statistics, we are not just interested in prediction accuracy. We also want to estimate the relation between observed predictor variables and the expectation of the response,

$$\mathrm{E}(Y|\mathbf{X}=\mathbf{x}) = F(\mathbf{x}). \tag{4.1}$$

{eq:exp-f}

In addition to using boosting for classification, like in the original AdaBoost, we would also like to use it in more general settings, and we therefore extend our discussion to a more general regression scheme where the outcome variable Y can be continuous. We are interested in interpreting the effects of the different covariates of $\mathbf{X}$ on the function $F(\cdot)$. A model for $F(\mathbf{X})$ which is amenable to such interpretation is the generalized additive model (GAM),

$$F(\mathbf{x}) = \alpha + \sum_{j=1}^{p} f_j(x_j), \tag{4.2}$$

{eq:gam}

where $\alpha \in \mathbb{R}$ is an offset and x_j is the j-th component of $\mathbf{x}$. $F(\mathbf{x})$ is a sum of component-wise functions f_j, and as such a GAM contains interpretable additive predictors. In 2000, Friedman et al. showed that AdaBoost fits a GAM with a forward stagewise algorithm, for a particular exponential loss function. (J. Friedman, Hastie, and Tibshirani 2000) This provided a way of viewing the successful boosting regime through a statistical lens. A year later, Friedman himself made a powerful insight for boosting. However, we must first discuss how we find an approximate solution for $F(\mathbf{X})$ in (4.1).

4.2 Finding a solution

We wish to estimate/approximate/find $F(\mathbf{X})$ in (4.1), so we are interested in solving

$$\underset{F}{\operatorname{argmin}} \operatorname{E}\left[\operatorname{L}(Y, F(\mathbf{X}))\right],$$

where L is an appropriate loss function. But as discussed in chapter 3 on Statistical learning theory, we have in practice observed a dataset $\{\mathbf{x}_i, y_i\}_{i=1}^{N}$, drawn from the joint distribution $(\mathbf{X}, Y)$. Therefore we substitute the loss expected with the expected risk (3.5), and so we want a solution to

$$\underset{F}{\operatorname{argmin}} \, \overline{\operatorname{err}}(F). \tag{4.3}$$

{eq:argmin-risk}

Finding such an F is not easy. We will now discuss a general optimization algorithm often used for this purpose.

Gradient descent

Gradient descent is an optimization algorithm for a differentiable multivariate function F. The motivation behind the gradient descent algorithm is that in a small interval around a point $\mathbf{x}$, F is increasing in the direction of the negative gradient at $\mathbf{x}$. Therefore, by moving slightly in that direction, F will increase. Indeed, with a sufficiently small step length, gradient descent will always converge, albeit to a local optimum. More formally, the algorithm is

1. Start with an initial guess $\mathbf{x}_0$, e.g. $\mathbf{x}_0 = \mathbf{0}$. Let $m = 1$.
2. Calculate the gradient $\mathbf{g}_{m-1} = -\nabla F(\mathbf{x}_{m-1})$.
3. Let $\mathbf{h}_m = \nu \mathbf{g}_{m-1}$, where ν is a small step length.
4. Let $\mathbf{x}_m = \mathbf{x}_{m-1} + \mathbf{h}_{m-1}$
5. Increase m, and go to step 2. Repeat until $m = M$.
6. The resulting final guess is $\mathbf{x}_M = \mathbf{x}_0 + \sum_{m=1}^{M} \mathbf{h}_m(\mathbf{x}_m)$

Back to our goal of finding an F to minimize (4.3). Often we choose a parameterized model $F(\mathbf{X}; \boldsymbol{\beta})$. Finding the optimal $\boldsymbol{\beta}$ analytically might be infeasible. The gradient descent algorithm can then be used. In this case, we fix $\mathbf{X}$ and let $F(\mathbf{X})$ in the algorithm be $F(\boldsymbol{\beta}; \mathbf{X})$. Thus we use gradient descent to find an optimal $\boldsymbol{\beta}$. We would then say we are doing gradient descent in parameter space. We are now ready to reveal Friedman's useful insight.

4.3 Gradient boosting: Functional gradient descent

There is another possible way to use gradient descent, and that is the important insight by Friedman in 2001. (J. H. Friedman 2001) He showed that instead of doing gradient descent in parameter space, one could do gradient descent in function space. Briefly, we first describe a naive way of doing this. Consider the function value at each $\mathbf{x}$ directly as a parameter, and use gradient descent directly on these parameters. However, this does not generalize to unobserved values $\mathbf{X}$, and we are after all interested in the population minimizer of (4.1). We can instead assume a parameterized form for F, e.g.,

$$F(\mathbf{X}; \{\boldsymbol{\beta}\}_{m=1}^{M}) = \sum_{m=1}^{M} \nu H(\mathbf{X}; \boldsymbol{\beta}_m), \tag{4.4}$$

{eq:gradboost}

where $H(\mathbf{X}; \boldsymbol{\beta})$ is also a function on the GAM form (4.2), but typically simpler, i.e., a base learner as discussed previously. We would like to minimize a data based estimate of the loss, i.e. the empirical risk, and so would choose $\{\boldsymbol{\beta}_m\}$ as the minimizers of

$$\underset{\{\boldsymbol{\beta}_m\}_{m=1}^{M}}{\operatorname{argmin}} \ \overline{\operatorname{err}}(H(\mathbf{x}; \{\boldsymbol{\beta}_m\})).$$

However, estimating these simultaneously may be infeasible. We can then use a greedy stagewise approach, where we at each step m choose the $\boldsymbol{\beta}_m$ which gives the best improvement, while not changing any of the previous $\{\boldsymbol{\beta}\}_{k=1}^{m-1}$. Hence at each step m the current solution is

$$F_m = F_{m-1} + \nu H(\mathbf{x}; \boldsymbol{\beta}_m),$$

where the parameters $\boldsymbol{\beta}_m$ are those in H minimizing the empirical risk when added to the fixed part F_{m-1}:

$$\boldsymbol{\beta}_m = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \overline{\operatorname{err}}(H(\mathbf{x}; \boldsymbol{\beta}_k) + H(\mathbf{x}; \boldsymbol{\beta})).$$

The final model is then the sum of these terms, like in (4.4). To find $\boldsymbol{\beta}_m$ in each step here, we use gradient descent. We have outlined a generic functional gradient descent algorithm. It can be stated more formally as follows.

1. Initialize $F_0(\mathbf{x})$, e.g., by setting it to zero for all components. Select a base learner H.

2. Compute the negative gradient vector,

$$U_i = -\frac{\partial \, \mathrm{L}(y_i, F_{m-1}(\cdot))}{\partial F_{m-1}(\cdot)}, \quad i = 1, \ldots, N.$$

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

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

4. Update $F_m(\cdot) = F_{m-1}(\cdot) + \nu H(\cdot; \boldsymbol{\beta}_m)$.

5. Repeat steps from 2 until $m = M$.

Note that while we call this functional gradient descent, this is exactly the gradient boosting algorithm.

4.4 L2Boost

In 2003, Buhlmann and Yu improve upon Friedman's work, and develop the L2Boost algorithm for which they also prove some important theoretical results (Bühlmann and Yu 2003). It is a special case of the generic functional gradient descent (FGD) algorithm, where we choose the squared error loss to be the loss function,

$$\mathrm{L}(y, F(\mathbf{x})) = \frac{1}{2} (y - F(\mathbf{x}))^2 .$$

The negative gradient vector of the loss then becomes the residual vector,

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

and hence the boosting steps become repeated refitting of residuals. (J. H. Friedman 2001, Bühlmann and Yu 2003). With $\nu = 1$ and $M = 2$, this had been proposed already in 1977 by Tukey, who called it "twicing". (Tukey 1977).

1. Initialize $F_0(\mathbf{x})$, e.g., by setting it to zero for all components. Select a base learner H, such as ordinary least squares, stumps, etc.

2. Compute the residuals

$$U_i = (y_i, F_{m-1}(x_i)), \quad i = 1, \dots, n$$

3. Estimate $\hat{H}_m$ by fitting $(\mathbf{X}_i, U_i)_{i=1}^{N}$ using the base learner H:

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

   Note that $\hat{H}(\cdot; \boldsymbol{\beta}_m)$ is then an estimate of the negative gradient vector.

4. Update $F_m(\cdot) = F_{m-1}(\cdot) + \nu H(\cdot; \boldsymbol{\beta}_m)$.

5. Repeat steps from 2 until $m = M$..

4.5 Component-wise gradient boosting

In high-dimensional settings, it might often be infeasible, if not impossible, to use a base learner H which incorporates all p dimensions. Indeed, using least squares base learners, it is impossible, since the matrix which must be inverted is singular when $p > N$. Component-wise gradient boosting is a technique/algorithm which does work in these settings. It was developed by Yu and Buhlmann (Bühlmann and Yu 2003), and it has further been refined and explored, e.g. by Buhlmann in Bühlmann 2006. The idea of the algorithm is to select p base learners. Each of these is only a function of the corresponding component of the data $\mathbf{X}$, i.e.,

$$h_j(\mathbf{x}) = h_j(x_j).$$

In each iteration, we fit all these learners separately, and choose only the one which gives the best improvement to be added in the final model. The resulting model $F_m(\cdot)$ is then a sum of componentwise effects,

$$F_m(\mathbf{X}) = \sum_{j=1}^{p} f_j(x_j),$$

where

$$f_j(x_j) = \sum_{m=1}^{M} \mathbb{1}_{mj} h_j(x_j; \boldsymbol{\beta}_m),$$

where $\mathbb{1}_{mj}$ is an indicator function which is 1 if component j was selected at iteration m and 0 if not. Hence this model is a GAM (4.2). Crucially, if we stop sufficiently early, we will typically perform variable selection. It is likely that some base learners have never been added to the final model, and as such those components in $\mathbf{X}$ are not added. We now give a presentation of the algorithm.

1. Start with an initial guess, e.g. $F_0 = \mathbf{0}$.
   Specify a set of base learners $h_1(\cdot), \dots, h_p(\cdot)$.
2. Compute the negative gradient vector $\mathbf{U}$.
3. Fit $(\mathbf{X}_i, U_i)_{i=1}^N$ separately to every base learner to get $\hat{h}_1(x_1), \dots, \hat{h}_p(x_p)$.
4. Select the component k which best fits the negative gradient vector.

$$k = \underset{j \in [1,p]}{\operatorname{argmin}} \sum_{i=1}^{N} (u_i - \hat{h}_j(\mathbf{x}_i))^2$$

5. Update $F_m(\cdot) = F_{m-1}(\cdot) + \nu h_k(x_k)$

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).

4.6 The importance of stopping early

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

Appendices

Bibliography

buhlmann2006 [1] Bühlmann, P. "Boosting for high-dimensional linear models". In: *Ann. Statist.* 34.2 (Apr. 2006), pp. 559–583.

buhlmann-yu [2] Bühlmann, P. and Yu, B. "Boosting With the L2 Loss". In: *Journal of the American Statistical Association* 98.462 (2003), pp. 324–339.

caroni2017 [3] Caroni, C. *First Hitting Time Regression Models*. John Wiley & Sons, Inc., 2017.

chhikara1988 [4] Chhikara, R. *The Inverse Gaussian Distribution: Theory: Methodology, and Applications*. Statistics: A Series of Textbooks and Monographs. Taylor & Francis, 1988.

adaboost [5] Freund, Y. and Schapire, R. E. "Experiments with a New Boosting Algorithm". In: *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*. ICML'96. Bari, Italy: Morgan Kaufmann Publishers Inc., 1996, pp. 148–156.

friedman2001 [6] Friedman, J. H. "Greedy function approximation: A gradient boosting machine." In: *Ann. Statist.* 29.5 (Oct. 2001), pp. 1189–1232.

friedman2000 [7] Friedman, J., Hastie, T., and Tibshirani, R. "Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors)". In: *Ann. Statist.* 28.2 (Apr. 2000), pp. 337–407.

kearnsvaliant [8] Kearns, M. and Valiant, L. G. "Crytographic Limitations on Learning Boolean Formulae and Finite Automata". In: *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*. STOC '89. Seattle, Washington, USA: ACM, 1989, pp. 433–444.

lee2010 [9] Lee, M.-L. T. and Whitmore, G. A. "Proportional hazards and threshold regression: their theoretical and practical connections". In: *Lifetime Data Analysis* 16.2 (Apr. 2010), pp. 196–214.

lee2006 [10] Lee, M.-L. T. and Whitmore, G. A. "Threshold Regression for Survival Analysis: Modeling Event Times by a Stochastic Process Reaching a Boundary". In: *Statist. Sci.* 21.4 (Nov. 2006), pp. 501–513.

mayr14a [11] Mayr, A. et al. "The Evolution of Boosting Algorithms. From Machine Learning to Statistical Modelling". In: *Methods of Information in Medicine* 53.6 (2014), pp. 419–427.

tukey [12] Tukey, J. *Exploratory Data Analysis*. Addison-Wesley series in behavioral science. Addison-Wesley Publishing Company, 1977.

threg

[13] Xiao, T. et al. “The R Package threg to Implement Threshold Regression Models”. In: *Journal of Statistical Software, Articles* 66.8 (2015), pp. 1–16.