Boosting the First-Hitting-Time Regression Model

Vegard Stikbakke

April 18, 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:

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

how to rephrase?

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.

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 $y_0 \notin \mathcal{B}$

what did you say about this again?

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

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

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.

```
1 library(threg)
2 data("lkr")
3 lkr$f.treatment2=factor(lkr$treatment2)
4 # head(lkr)
5 fit <- threg(Surv(weeks, relapse) ~ f.treatment2|f.treatment2,
      data=lkr)
6 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 boosting

Boosting

Boosting is one of the most promising methodological approaches for data analysis developed in the last two decades. (Mayr et al. 2014) The history of boosting started with the question posed in 1989 by Kearns and Valiant, working on computational learning theory, of whether any weak learner could be transformed to become also a strong learner. (Kearns and Valiant 1989) A weak classifier is in general defined to be one which is only slightly better than random choice. For regression, it is a bit harder to give a specific definition, but a weak regressor is simple and low dimensional, and does not pick up much of the underlying signal. The answer to the original question is yes, and Schapire and Freund showed this with the AdaBoost algorithm, which constructs a binary classifier. (Freund and Schapire 1996) The algorithm works by iteratively reweighting observations, giving more weight to misclassified observations, and training a new base learner on all observations, using the updated weights. The resulting AdaBoost classifier is a linear combination of these base 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.

3.1 Statistical boosting

sec:sboost

In statistics, however, we are interested in models which are interpretable. We want to estimate the relation between observed predictor variables and the expectation of the response,

$$\mathrm{E}(Y|X = x) = f(x).$$

In addition to using boosting for classification, like in the original AdaBoost, we would also like to use it in more general settings. We therefore extend our discussion to the more general regression scheme, where the outcome variable Y can be continuous. To evaluate a candidate $\hat{f}(x)$, we need to see how well it estimates $f(x)$. This is typically done by choosing a loss function,

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

{eq:loss}

and calculating the empirical risk, i.e., the average in-sample error over some observed test data set. A typical loss function for regression is the L_2 loss,

$$\mathrm{L}(Y, f(X)) = (Y - f(X))^2$$

The empirical risk is then

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

A possible model for $f(x)$ is the generalized additive model (GAM), in which different effects of single predictors are added,

$$f(x) = \beta_0 + \sum_{i=1}^{p} h_p(x_p). \tag{3.2}$$

{eq:gam}

In 2000, Friedman 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.

Gradient boosting

Gradient boosting was proposed in 2001 (J. H. Friedman 2001), and further refined in 2003 (Bühlmann and Yu 2003). We will here explain the gradient boosting framework, starting with the motivation in the gradient descent algorithm.

Gradient descent

Gradient descent, or steepest descent, is a greedy iterative algorithm for numerically maximizing a differentiable objective function F. At each iteration step it improves on the previous solution by going in the direction which increases the objective function the most. This is by definition the direction of the gradient. For our purposes, the objective function is the negative loss function, since we are *minimizing* the loss. To avoid going too far, i.e., beyond the closest local optima, we choose a small step size ν, and it has been shown empirically that $\nu = 0.1$ is a good choice (Bühlmann and Yu 2003, Bühlmann 2006). The gradient descent search stops when reaching a maximum, possibly a local maximum. The result of the search after M steps is a sequence of improvements, or boosts, on the objective function.

Setting

Assume we have data $\mathbf{X} \in \mathbb{R}^p$ and $Y \in \mathbb{R}$ with some relation $Y = F(\mathbf{X})$, $F\colon \mathbb{R}^p \to \mathbb{R}$, which we wish to estimate. We have the relationship

$$Y = F(\mathbf{X}) + \varepsilon,$$

where ε is a random variable with expectation zero. We wish to find a $\hat{F}(\cdot)$ which minimizes the expected loss between it and the distribution $Y = F(\cdot)$,

$$F^* = \min \mathrm{E}_{Y,\mathbf{X}}\left[\mathrm{L}(Y, \hat{F}(\mathbf{X}))\right] = \min \mathrm{E}_Y\left\{\mathrm{E}_{\mathbf{X}}\left[\mathrm{L}(Y, \hat{F}(\mathbf{x}))|\mathbf{X} = \mathbf{x}\right]\right\}, \tag{3.3}$$

{eq:min-loss}

where L is some meaningful loss function which measures the difference between Y and $F(\mathbf{X})$. In particular, a loss function is 0 if Y is exactly equal $F(\mathbf{X})$, and positive otherwise. To estimate F we typically choose a parameterized model,

$$F(\mathbf{X}) = H(\mathbf{X}; \boldsymbol{\gamma}), \tag{3.4}$$

where $H : \mathbb{R}^p \to \mathbb{R}$ is some function of parameters $\boldsymbol{\gamma}$, which are to be estimated. In other words, we are trying to solve

$$\boldsymbol{\gamma}^* = \underset{\boldsymbol{\gamma}}{\operatorname{argmin}} \, \mathrm{E}_Y \left\{ \mathrm{E}_{\mathbf{X}} \left[\mathrm{L}(Y, H(\mathbf{x}; \boldsymbol{\gamma})) | \mathbf{X} = \mathbf{x} \right] \right\}. \tag{3.5}$$

{eq:}

In practice, for finite observed data points $\{\mathbf{x}_i, y_i\}_{i=1}^N$, we must estimate the expected loss (3.3) by the empirical risk,

$$\hat{L}(y, h(x; \boldsymbol{\gamma})) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, H(\mathbf{x}_i; \boldsymbol{\gamma})). \tag{3.6}$$

{eq:emp-risk}

For finite data points and a chosen such model H, there exists a $\boldsymbol{\gamma}^*$ which minimizes (3.3),

citation needed?

$$\boldsymbol{\gamma}^* = \min_{\boldsymbol{\gamma}} R(H; \boldsymbol{\gamma}) = \min_{\boldsymbol{\gamma}} \frac{1}{N} \sum_{i=1}^{N} L(y_i, H(\mathbf{x}_i; \boldsymbol{\gamma})) \tag{3.7}$$

{eq:min-loss-param}

but estimating this is not necessarily easy. An algorithm which is often used to find an approximate solution $\hat{\boldsymbol{\gamma}}$ to 3.7 is gradient descent in parameter space.

Gradient descent in parameter space

At iteration step m we find an improvement $\boldsymbol{\beta}_m$, and the current solution at this step is $\boldsymbol{\gamma}_m = \sum_{j=0}^{m} \boldsymbol{\beta}_j$. We start with an initial guess $\boldsymbol{\beta}_0$, say $\boldsymbol{\beta}_0 = \bar{y}$. We then carry out steps $m = 1, \ldots, M$, where we find increments $\boldsymbol{\beta}_m$ which improve our existing solution $\boldsymbol{\gamma}_{m-1}$, using gradient descent on the parameters. Hence at each iteration, we compute the gradient of the loss with respect to the parameters, evaluated at the current solution,

$$\mathbf{g}_m = \{g_{jm}\} = \left\{ \frac{\partial}{\partial \gamma_{m-i,j}} \frac{1}{N} \sum_{i=1}^{N} L(y_i, H(\mathbf{x}_i; \boldsymbol{\gamma}_{m-1})) \right\}_{j=1}^{p}. \tag{3.8}$$

We choose $\boldsymbol{\beta}_m = \nu \mathbf{g}_m$, for step size ν, and iterate.

Gradient descent in function space

We have until now viewed the function estimating problem as the problem of minimizing a function by optimizing its parameters, i.e., doing gradient descent in parameter space. We can instead view the optimization problem (3.3) from a non-parametric perspective, by considering $F(\mathbf{x})$ at each point $\mathbf{x}$ as a parameter. But doing this directly is not helpful, for we wish to estimate the underlying function, which takes values different from the observed $\mathbf{x}$'s. We therefore assume a parameterized form such as

$$F(\cdot; \{\boldsymbol{\beta}_m\}_{m=1}^{M}) = \sum_{m=1}^{M} \nu H(\cdot; \boldsymbol{\beta}_m), \tag{3.9}$$

where H is a function parameterized by $\boldsymbol{\beta}_m$ and again ν is a (typically small) step size. Hence F is an additive basis expansion, where $H(\cdot; \boldsymbol{\beta}_m)$ is the family of basis functions. Minimizing this may be infeasible, since it involves simultaneously optimizing several functions and several parameters. In such situations one can try a stagewise approach, at each iteration m choosing $H(\mathbf{x}; \boldsymbol{\beta})$ such that it gives the best improvement, while not changing the previous $m-1$ functions. Following the numerical optimization paradigm as above, we take our approximate solution $\hat{F}$ to be the sum of a sequence of improvements, or boosts,

$$\hat{F}_M = \sum_{m=0}^{M} \nu H_m(\cdot; \boldsymbol{\beta}_m). \tag{3.10}$$

L_2Boost with OLS regression

Let us look at a more concrete example. Assume data $(\mathbf{X}_i, Y_i)_{i=1}^N$, where $\mathbf{X}_i \in \mathbb{R}^2$. We use the L_2 loss function,

$$L(x, y) = \frac{1}{2} \left(y - f(x)\right)^2, \tag{3.11}$$

{l2}

such that the derivative of L is simply the residual $(y - f(x))$. We also use basis functions $h(\mathbf{X}; \hat{\boldsymbol{\beta}}_m) = \alpha + \boldsymbol{\beta}_m^T \mathbf{X}$, i.e., ordinary linear regression. At each step the $\hat{\boldsymbol{\beta}}$ are the regular least squares minimizers. With the L_2 loss, the gradient boosting algorithm then becomes as follows. Start with an initial guess $h_0(\mathbf{X}, Y) = h(\mathbf{X}, Y; \hat{\boldsymbol{\beta}}_0)$, i.e., one ordinary least squares regression fit. Then $F_0(\mathbf{X}) = h_0(\mathbf{X})$. Then, for $m = 1, \ldots, M$, calculate the gradient of the loss function, i.e., the residuals,

$$U_i = Y_i - F_{m-1}(\mathbf{X}_i). \tag{3.12}$$

{residuals}

Then fit the base learner h to the residuals, i.e.,

$$h_m(\mathbf{X}_i, U_i) = \mathbf{X}_i (\mathbf{X}_i^T \mathbf{X}_i)^{-1} \mathbf{X}_i^T U_i, \tag{3.13}$$

and add it to the final model,

$$F_m = \sum_{j=0}^{m} h_m. \tag{3.14}$$

Component-wise gradient boosting

If $N < p$, it may be infeasible to use base learners which use all p dimensions. In particular, in OLS regression, we would need to calculate a matrix-vector product, which would be impossible, because it would give a singular matrix. One way to solve this is to use base learners which only incorporate one dimension at a time. Bühlmann and Yu 2003 developed this algorithm using component-wise smoothing splines as base learners. They concluded that in high dimensions the $L2$Boost algorithm performs as well as boosted trees. At each iteration, one then finds the best base learner for each dimension, and adds the best of these to the final model. Hence at each iteration, only one (regularized) predictor is added. A clear advantage this is that if one stops boosting early enough, this will often do variable selection (!), meaning many predictors will not be included in the final model F.

3.2 Statistical 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

$$\mathrm{L}(Y, \hat{F}(\mathbf{X})). \tag{3.15}$$

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

$$\mathrm{L}(Y, \hat{F}(\mathbf{X})) = (Y - \hat{F}(\mathbf{X}))^2. \tag{3.16}$$

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

$$\mathrm{Err}_\tau = \mathrm{E}[\mathrm{L}(Y, \hat{F}(\mathbf{X}))|\tau], \tag{3.17}$$

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

$$\mathrm{Err} = \mathrm{E}[\mathrm{Err}_\tau] = \mathrm{E}_\tau \left([\mathrm{L}(Y, \hat{F}(\mathbf{X}))|\tau] \right). \tag{3.18}$$

{eq:err}

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

$$\overline{\mathrm{err}} = \frac{1}{N} \sum_{i=1}^{N} \mathrm{L}(y_i, \hat{F}(\mathbf{x}_i)), \tag{3.19}$$

also known as the empirical risk. To estimate err (3.18), 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

$$\hat{\mathrm{Err}} = \frac{1}{M} \sum_{i=1}^{M} L(y_i, \hat{F}(\mathbf{x}_i)), \tag{3.20}$$

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

3.3 Boosting

We will now discuss boosting which is one of the most promisiing methodological approaches developed in the latest ... years.

AdaBoost...

Statistical view of boosting

In addition to hopefully finding an $\hat{F}(\cdot)$ which minimizes the test/generalization error Err, we are interested in interpreting the effects of the different covariates of $\mathbf{X}$ on the fitted function $\hat{F}(\cdot)$. A model which is amenable to such interpretation is the generalized additive model (GAM),

$$F(\mathbf{x}) = \alpha + \sum_{j=1}^{p} f_j(x_j), \tag{3.21}$$

where x_j is the j-th component of $\mathbf{x}$. We see this is a component-wise functon for each component, or the sum of component-wise f's.

We are interested in finding the best f,

$$F^* = \underset{F}{\operatorname{argmin}} \operatorname{Err}(f). \tag{3.22}$$

Gradient descent

An optimization algorithm for a differentiable multivariate function F. The motivation behind gradient descent 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. Initialize x_0 with an initial guess, e.g. $x_0 = 0$. Let $m = 1$.
2. Calculate $-\nabla F(x_{m-1}) = \mathbf{g}_m(\mathbf{x}_{m-1})$.
3. Let $\mathbf{x}_m = \mathbf{x}_{m-1} + \nu \mathbf{g}(\mathbf{x}_{m-1})$, where ν is a small step length.
4. Increase m, and go to step 2. Repeat until convergence.
5. Resulting final guess is $\hat{\mathbf{x}} = \mathbf{x}_0 + \nu \sum_{m=1}^{M} \mathbf{g}_m(\mathbf{x}_m)$

Gradient boosting

The gradient boosting can very well be used to find optimal parameters of a function $H(\mathbf{X}; \boldsymbol{\beta})$, such that in the gradient descent algorithm we use $F(\boldsymbol{\beta}) = H(\dots; \boldsymbol{\beta})$ and find an optimal $\hat{\boldsymbol{\beta}}$. We would then say we are doing gradient descent in parameter space. This is quite possible and a good idea if the optimal parameters of H are hard to find. There is another possible way to use gradient descent, and this algorithm and the important insight therein was worked out by Friedman in 2001. (J. H. Friedman 2001) He argued that one could instead do gradient descent in function space. A naive way of doing this is to consider the function value at each $\mathbf{x}$ directly as a parameter. However this does not generalize to unobserved values $\mathbf{X}$. We can instead assume a parameterized form, e.g.,

$$F(\mathbf{X}; \{\boldsymbol{\beta}\}_{m=1}^{M}) = \sum_{m=1}^{M} \nu H(\mathbf{X}; \boldsymbol{\beta}_m). \tag{3.23}$$

{eq:gradboost}

We would like to minimize a data based estimate of the expected loss (the empirical risk), and so would choose $\{\boldsymbol{\beta}_m\}$ as

$$\{\boldsymbol{\beta}_m\}_{m=1}^{M} = \underset{\boldsymbol{\beta}'_m}{\operatorname{argmin}} \sum_{i=1}^{N} \mathrm{L}\left(y_i, \nu \sum_{m=1}^{M} h(\mathbf{x}; \boldsymbol{\beta}'_m)\right). \tag{3.24}$$

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

$$\boldsymbol{\beta}_m = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \sum_{i=1}^{N} L\left(y_i, \nu \left[\sum_{k=1}^{m-1} h(\mathbf{x}; \boldsymbol{\beta}_k) + h(\mathbf{x}; \boldsymbol{\beta})\right]\right), \tag{3.25}$$

and

$$F_m = F_{m-1} + \nu h(\mathbf{x}; \boldsymbol{\beta}_m). \tag{3.26}$$

The final model is then the sum of these terms, like in (3.23). To find $\boldsymbol{\beta}_m$ in each step here, we might use gradient descent. This, then, is gradient boosting. A generic functional gradient descent is as follows.

1. Initialize $F_0(\mathbf{x})$, e.g., by setting it to zero for all components.
2. Compute the negative gradient vector,

$$U_i = -\frac{\partial \mathrm{L}(y_i, F_{m-1}(\cdot))}{\partial F_{m-1}(\cdot)}, i = 1, \ldots, N. \tag{3.27}$$

3. Estimate $\hat{h}_m$ by fitting $(\mathbf{X}_i, U_i)$ using a base learner h:

$$\boldsymbol{\beta}_m = \operatorname{argmin} \boldsymbol{\beta} \sum_{i=1}^{N} \mathrm{L}(u_i, h(\mathbf{x}_i; \boldsymbol{\beta})) \tag{3.28}$$

   $\hat{h}(\cdot; \boldsymbol{\beta}_m)$ is then an estimate of the negative gradient vector (!).
4. Repeat above steps.
5. Update $F_m(\cdot) = F_{m-1}(\cdot) + \nu h(\cdot; \boldsymbol{\beta}_m)$.

L2Boost

L2Boost is an algorithm which was developed by Buhlmann and Yu in 2006 (Bühlmann and Yu 2003), and it is a special case of the generic functional gradient descent (FGD) algorithm. In it, we use the squared error loss as the loss function in the FGD algorithm,

$$L(y, \hat{F}(\mathbf{x})) = \frac{1}{2} \left(y - \hat{F}(\mathbf{x}) \right)^2 \tag{3.29}$$

The negative gradient vector then becomes the residual vector, 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 in 1977 by Tukey, as "twicing". (Tukey 1977).

We will outline the algorithm here.

Gradient boosting is functional gradient descent.

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 in the same paper (Bühlmann and Yu 2003), and has further been refined and explored. 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 $\mathbf{X}$,

$$h_j(x_j). \tag{3.30}$$

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. Tje resulting model $F_m(\cdot)$ is then a sum of componentwise effects,

$$F_m(\mathbf{X}) = \sum_{j=1}^{p} f_j(x_j), \tag{3.31}$$

where

$$f_j(x_j) = \sum_{m=1}^{M} h_j(x_j; \boldsymbol{\beta}_m). \tag{3.32}$$

This model is a GAM. 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. Initialize. $m = 0$, $F_0 = \mathbf{0}$, *e.g.*. Specify a set of base learners $h_1(x_1), \ldots, h_p(x_p)$.
2. Compute the negative gradient vector u.
3. Fit u separately to every base learner.
4. Select component k which best fits the negative gradient vector.

$$k = \underset{j \in [1,p]}{\operatorname{argmin}} \sum_{i=1}^{N} (u_i - h_j(x_i))^2 \tag{3.33}$$

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

In fact, ... and ... argue that boosting is appropriate not in low or medium dimensions, but in high dimensions.

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.