Modeling the Exposure and the Outcome: eXtreme Gradient Boosting

Ashley I Naimi

June 2022

Contents

1 Gradient Boosting: Basic Introduction

Gradient boosting is our first introduction to meta-learning. The basic idea behind meta-learning is to combine several "weak" learners into a single algorithm. These "weak" learners are sometimes referred to as stage 0 learners, and the meta learner is sometimes referred to as the stage 1 learner. In theory, the overall performance of the stage 0 learners, when combined (i.e., the stage 1 learner) should be greater than the performance of any stage 0 learner by itself. There are many ways to combine stage 0 learners into stage 1 (meta) learners. These include bootstrap aggregation (bagging), stacking (super learner), and boosting.

Gradient boosting is best understood with a simple model $m_1(X)$ and mean squared error as the loss function. Suppose we fit this model to some dataset, with outcome Y and a set of covariates X. Suppose we use ordinary least squares to do this, with the following model:

$$m_1(X) = E(Y \mid X) = \alpha_0 + \alpha_1 X_1 + ... + \alpha_p X_p$$

With an estimate $\hat{m}_1(X)$ of the model, we can compute residuals:

$$\hat{r}_1(X) = Y - \hat{m}_1(X)$$

If the original model $m_1(X)$ was estimated with mean squared error as the loss function (say, using ordinary least squares), then these residuals are equivalent to the gradient for the model. We can then use "gradient boosting" and fit another model to the residuals:

$$m_2(X) = E[\hat{r}_1(X) \mid X] = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p$$

We can then "boost" the first model by updating it with the information from the second model (there are several ways to do this, often with the inclusion of hyperparameters, such as a "learning rate". See chapter 10 of Hastie et al. (2009) or section 8.2.3 or James et al. (2017)). This "boosting" process can occur iteratively, until a prespecified number of iterations is reached, or a performance threshold attained.

Note that one need not use a linear regression model with OLS for boosting to work. In fact, it is common to implement gradient boosting with classifica-

tion and regression trees (Hastie et al., 2009, section 10.9).

2 Extreme Gradient Boosting

Extreme gradient boosting, and its common software implementation, XGBoost, was introduced in 2016 as a more regularized from of gradient boosting. There are a number of ways to implement XGBoost (e.g., the standard way, via caret, via the SuperLearner, other). Here is an example implementation using the `caret` package.

```
library(tidyverse)
library(xgboost)
library(caret)
library(DiagrammeR)

nhefs <- read_csv(here("data", "nhefs.csv")) %>%
    mutate(wt_delta = as.numeric(wt82_71 >
        median(wt82_71)))

head(nhefs)
```

```
## # A tibble: 6 x 12
##    seqn  qsmk   sex   age income   sbp   dbp price71 tax71  race wt82_71
##   <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>   <dbl> <dbl> <dbl>   <dbl>
## 1   233     0     0    42     19   175    96    2.18 1.10      1  -10.1
## 2   235     0     0    36     18   123    80    2.35 1.36      0    2.60
## 3   244     0     1    56     15   115    75    1.57 0.551     1    9.41
## 4   245     0     0    68     15   148    78    1.51 0.525     1    4.99
## 5   252     0     0    40     18   118    77    2.35 1.36      0    4.99
## 6   257     0     1    43     11   141    83    2.21 1.15      1    4.42
## # ... with 1 more variable: wt_delta <dbl>
```

```
# create tuning grid
grid_default <- expand.grid(nrounds = c(250),
    max_depth = c(4), eta = c(0.01), gamma = c(0,
```

```
        1), min_child_weight = c(10, 25),
    colsample_bytree = c(0.7), subsample = c(0.6))
# set random seed
set.seed(123)
# train XGBoost model
xgboost1 <- train(factor(wt_delta) ~ qsmk +
    sex + age + income + sbp + dbp + price71 +
    tax71 + race, data = nhefs, tuneGrid = grid_default,
    method = "xgbTree", metric = "Kappa")

tree_plot <- xgb.plot.tree(model = xgboost1$finalModel,
    trees = 1:2, plot_width = 2000, plot_height = 2000)
```

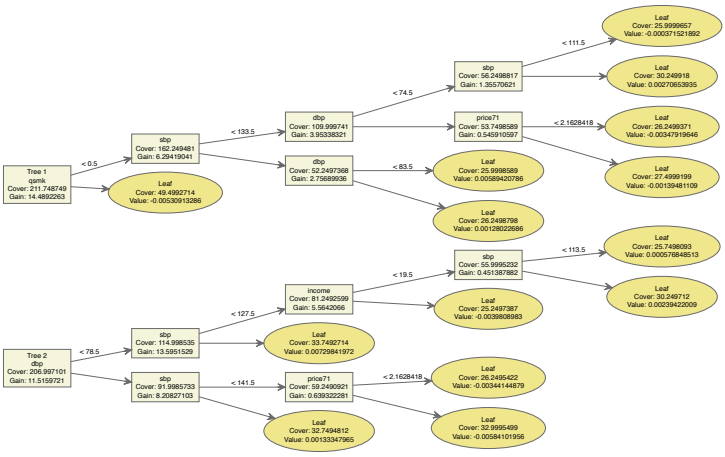

Figure 1: Two trees from the example extreme gradient boosting algorithm fit to the NHEFS data

XGBoost is a very popular algorithm, with a wide variety of tuning parameter options for improving the performance of the model. Here, we explore a handful:

2.1 nrounds (aka ntrees)

This parameter is similar to the random forest parameter. It determines the number of trees in the gradient boosting forest. We will become familiar with this in the section on random forests. Increasing the number of trees in the forest can slow down the algorithms fit considerably. One range I often explore

100 and 1000 trees.

2.2 max_depth

The depth of a tree corresponds to how many branches deep each tree is allowed to go. A larger depth corresponds to more flexible trees, but this can also lead to overfitting. The default tree depth in `xgboost` is 6, and it is often useful to explore `max_depth` range of between 4 and 8.

2.3 eta (aka shrinkage)

The eta parameter controls the manner in which the boosted tree at a given iteration is merged with the tree in the previous iteration. In the example above, we noted that the $\hat{m}_1(X)$ model is "boosted" by getting updated with the residual fit with the $\hat{m}_2(X)$ model. This boosting usually occurs through some function, where the residual contribution is penalized or regularized through some value eta. This parameter tends to have an important effect on the fit of the model. The default value is 0.3, and it is often important to evaluate a range of effects between 0.01 and 0.5.

2.4 min_child_weight (aka minobspernode)

This parameter refers to the minumum allowable number of observations in each node. The default for this value is 1. However, this number is quite low, and caution is warranted here, particularly when the max_depth is large. I always tend to set this value to at least ten, and will often explore ranges between 10 and 50.

2.5 gamma

The gamma parameter controls the extent to which changes in the tree structure occur on the basis of changes in the loss function. If a particular split in the tree will only lead to a small change in the loss function, the split may not be worth it. The gamma parameter defines what it means to be "worth it". The default gamma value is 0.

3 Final Note

It's important to note that there is never a single optimal tuning parameter choice for all (or even most) settings. In fact, the choice of a given tuning parameter often has important effects on the optimal value of other tuning parameters. There are usually important tradeoffs to consider between tuning parameter values. For example, in the context of XGBoost, a high value of `min_child_weight` may require a lower `eta` or `max_depth` value. However, this is true for other types of machine learning algorithms.

References

Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, NY, 2009.

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer, New York, 2017.