Project 2 - Gradient Boosting Trees

Willie Langenberg

12/8/2021

Task 1

a)

I decided to use the function gbm from the gbm package in R. The package delivers functions with implementations of various gradient boosting models extended from Freund and Shapire's AdaBoost Algorithm (1997) and J. Friedman's gradient boosting machine (2001).

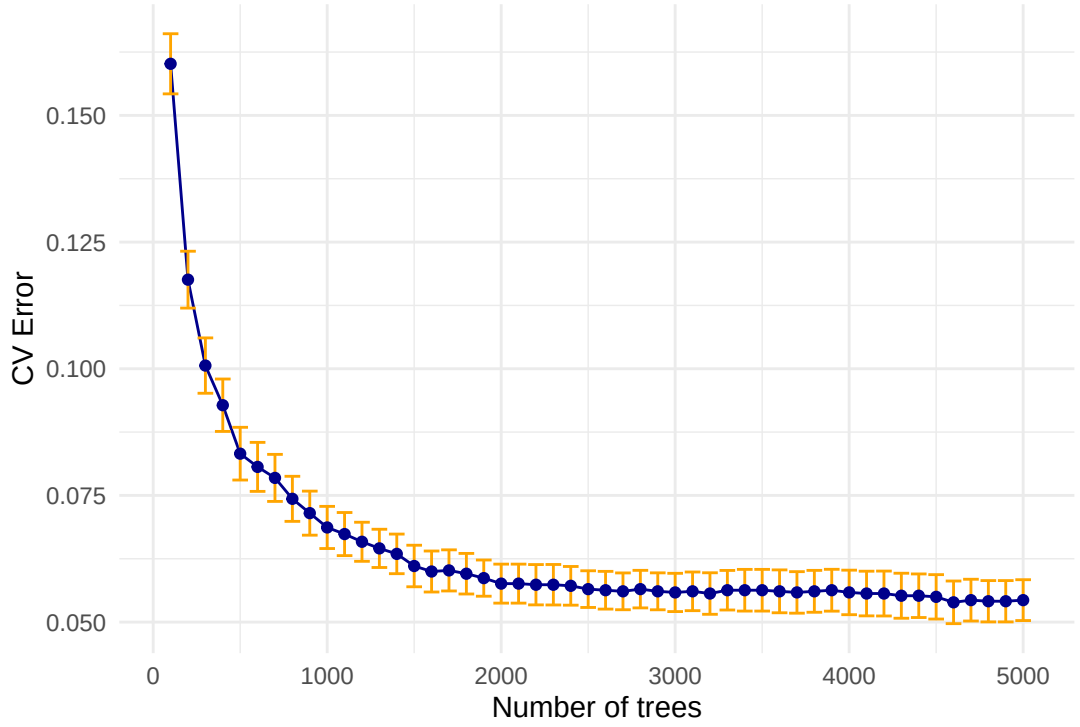
First of all you can specify the formula and data as with any other statistical modelling function in R. Formula is where you tell how we want to model our data, so which column to use as response variable and which are the predictors. Further, since we are dealing with a binary classification problem, I will choose bernoulli as distribution for the (gbm) function. Also I will specify that we will use 1500 trees by setting the argument n.trees to 1500. I want each tree to consist of only a stump, specified by setting the interaction.depth to 1. For each iteration of the boosting algorithm I want the model to learn slowly since we are using quite a lot of trees and this is done by changing the shrinkage argument. I set it to the value 0.01.

You are also able to specify arguments such as weights, for the fitting process, but I leave this as default (empty). Further n.minobsinnode specify the minimum amount of observation in a terminal node, I set this as zero because I do not have a preference about this. Then there is var.monotone to specify if we know that the predictors have any monotone relationship with the response variable, which I have no knowledge about, hence I leave it as default (NULL). Also, we have the option to specify bag.fraction to include some randomness to the model in the way of only using a fraction of the data to train next tree in the algorithm, I leave this at default 0.5 since I do not really have a preference about this either. Lastly we can specify if we want to only train on a fraction of the data, and save some for estimating the loss function on out-of-sample data, I will not do this. Also, we can specify cross validation folds (default = 0), stratify classes (default = NULL), choose how many CPU cores (I use 8) to use and choose if we want to print performance outputs (verbose, with default = FALSE).

b)

See appendix for code to construct gradient boosting trees with different number of nodes. I added arguments to be able to specify amount of nodes, trees and how many folds to use with cross validation.

c)



By using the code written in b) we can now construct the process of training and also estimating the missclassification rate using cross-validation, and then continue to plot our results. I created a function called crossvalidation_plot where you can specify the maximum amount of trees to test, and the amount of steps to take in that direction, i.e. test $[100, 200, 300, ..., 5000]$ would be crossvalidation_plot(M = 5000, step = 100). The function iterates over the different amount of trees to train the model by, and performs a cross-validation estimate on each iteration. So we have a vector of 10 CV estimates in each iteration, in which we save the mean and standard error of the CV estimates. With these we can construct the plot, having mean CV estimate on the y axis with corresponding standard error, and amount of trees on the x axis. From this results I could find that the lowest CV estimate for 4600 trees at 0.0539 CV error and 0.00421 standard error. The CV estimate with lowest amount of trees, within one standard error of 0.0539 is with 2000 trees at 0.0576 CV error.

d)

Since I am using a low shrinkage paramter and low depth of the individual trees I need more trees to learn from the data. So we can see that accoring to our plot in 1c) the cross-validation error is decreasing when the number of trees increases. With connection to the bias-variance tradeoff we are getting an lower bias, but will get an increasing variance if we are not careful. Our goal is generally to get as low bias and variance as possible, and I assume that the standard-error-rule suggests a relatively good trade between the bias and variance.

Task 2

a)

The K-class multinomial deviance loss function is given by

$$L(y, p(x)) = -\sum_{k=1}^{K} I(y = G_k) log\ p_k(x) = -\sum_{k=1}^{K} I(y = G_k) f_k(x) + log\big(\sum_{\ell=1}^{K} e^{f_\ell(x)}\big).$$

We are to calculate the gradient of the loss function

$$\partial L(y_i, f(p_i))/\partial f(p_i).$$

b)

We want to derive the formula for the variance of the average of B identically distributed random variables X (with positive pairwise correlation ρ) given by

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.$$

So let

$$S = \sum_i \frac{X_i}{B},$$

Then the variance of S is followed by

$$Var(S) = \frac{1}{B^2}\bigg(\sum Var(X_i) + \sum_{i \neq j} Cov(X_i, X_j)\bigg)$$

which is equal to (since dependency between X_i and X_j, and identical distributions)

$$Var(S) = \frac{1}{B^2}\bigg(B\sigma^2 + \sum_{i \neq j} \rho\sigma^2\bigg) = \frac{1}{B^2}\bigg(B\sigma^2 + \rho\sigma^2(B^2 - B)\bigg)$$

which simplifies to

$$Var(S) = \frac{\sigma^2}{B} + \rho\sigma^2 - \frac{\rho\sigma^2}{B} = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.$$

This is the expression we wanted to show. We can see that if ρ is negative we could get negative variance for large values of B, which is not possible. I tried to see in what step, or why this happened but I could not figure it out.

c)

So by changing the hyperparameter m we are essentially limiting the amount of information each tree can hold. With a larger m I assume we are increasing the model complexity leading to higher variance but lower bias. If we instead choose a lower m, and restricting each tree to only a smaller subset of the p variables we would probably get a lower variance but higher bias.

Appendix

```r
# Loading packages used for later analysis
library(tidyverse)
library(gbm)

# Reading in the data.
setwd("~/Documents/Statistical Learning/Project 2/")
url <- "Data/data.txt"
df <- read.table(url)


# Task 1 - a) Fitting a gradient boosting tree:
model_1 <- gbm(V58 ~ ., data=df, distribution = "bernoulli", n.trees = 1500, interaction.depth = 1, n.m

# Task 2 - b) Construct gradient boosting trees with different number of nodes.
gradient_boosted_trees_node <- function(data_df, nodes, trees=1500, folds=0) {
  boosted_model <- gbm(V58 ~ ., data=data_df, distribution = "bernoulli", n.trees = trees, interaction.
  return(boosted_model)
}

### Stump
stump_model <- gradient_boosted_trees_node(df, 1)
### 5
node_5_model <- gradient_boosted_trees_node(df, 5)
### 10
node_10_model <- gradient_boosted_trees_node(df, 10)
### 20
node_20_model <- gradient_boosted_trees_node(df, 20)
### etc...

# Task - c) Cross-validation

missclassification_error <- function(x, y) {
  return (sum(x!=y$V58)/length(x))
}

cv_error_est <- function(x) {
  return (c(mean(x),sd(x)/sqrt(length(x))))
}

crossvalidation_errors <- function(data=df, M=100, step=50) {
  # Starting by shuffle the data
  df_shuffled <- df[sample(nrow(df)),]
  # Creating 10 folds, to use on all models
  cv_folds <- cut(seq(1,nrow(df_shuffled)), breaks=10, labels=FALSE)
  #Variable to store results for each fold
  error_rates <- NULL
  # For each fold, train on 9 folds and use the remaining as test set.
  for(i in 1:10){
    print(c('Fold:', i))
    # Splitting data accoring to cross validation folds
    kfold_index <- which(cv_folds == i,arr.ind=TRUE)
    testData <- df_shuffled[kfold_index, ]
    trainData <- df_shuffled[-kfold_index, ]
```

```
    # Train a gradient boosted model with only a stump, interaction depth = 1, trees = m, and with 10-f
    model <- gradient_boosted_trees_node(trainData, nodes = 1, trees = M, folds=0)
    #predict using different amount of trees
    preds <- predict.gbm(model, testData, seq(step, M, step))
    # Convert predictions to classes (0,1)
    preds_class <- apply(preds>0, FUN = as.integer, 2)
    # Miss classification on test data for each fold
    missclass_error_rate <- apply(preds_class, 2, FUN = missclassification_error, y = testData)
    error_rates <- rbind(error_rates, missclass_error_rate)
  }
  # Calculate mean and standard error for the cv estimated errors.
  output <- apply(error_rates, 2, FUN = cv_error_est)
  output <- as_tibble(cbind(c(output[1,]), c(output[2,]))) %>%
    mutate("x" = seq(step, M, step)) %>%
    rename(mean = V1, se = V2)

  return(output)
}

cv_error_est <- crossvalidation_errors(df, 5000, 100)

cv_error_est %>% ggplot(aes(x=x, y=mean)) +
  theme_minimal() +
  geom_line(color="darkblue") +
  geom_errorbar(aes(ymin=mean-se, ymax=mean+se), color="orange") +
  geom_point(color="darkblue") +
  xlab("Number of trees") +
  ylab("CV Error")

# Lowest CV (4600 number of trees)
lowest_cv <- cv_error_est[which.min(cv_error_est$mean), ]
lowest_cv_se <- lowest_cv$mean + lowest_cv$se

# CV within 1 standard error (2000 trees)
cv_error_est[cv_error_est$mean<lowest_cv_se,][1,]
```