PS3 Borui Sun

borui sun

2/15/2020

Decision Trees

1. Set up the data and store some things for later use:
    - Set seed
    - Load the data
    - Store the total number of features minus the biden feelings in object `p`
    - Set λ (shrinkage/learning rate) range from 0.0001 to 0.04, by 0.001

```
nes2008 <- read_csv("./data/nes2008.csv")
p <- sum(!grepl("biden", colnames(nes2008)))
lambda <- seq(0.0001, 0.04, by = 0.001)
```

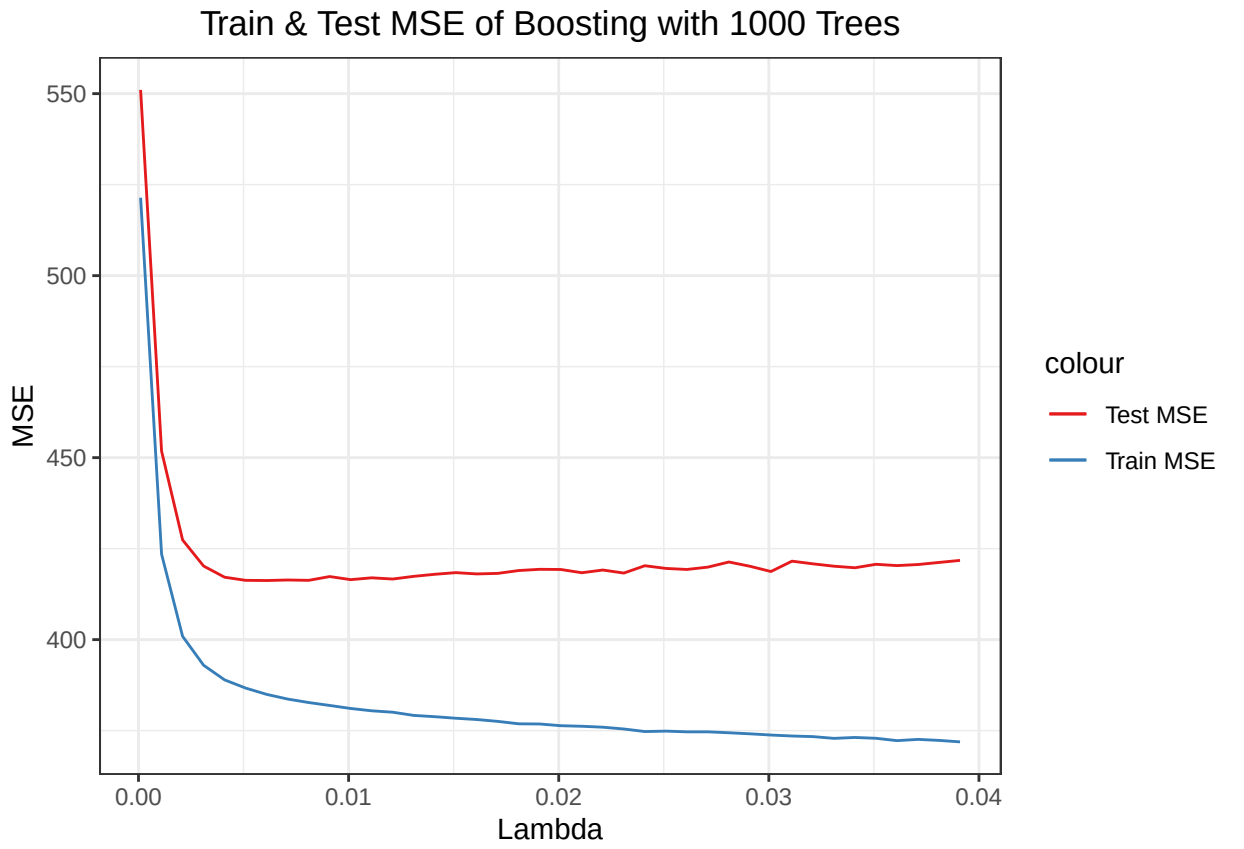
2. (10 points) Create a training set consisting of 75% of the observations, and a test set with all remaining obs. **Note**: because you will be asked to loop over multiple λ values below, these training and test sets should only be integer values corresponding with row IDs in the data. This is a little tricky, but think about it carefully. If you try to set the training and testing sets as before, you will be unable to loop below.

```
set.seed(123)
split <-initial_split(nes2008, prop = .75)
train <- training(split)
test <- testing(split)
```

3. (15 points) Create empty objects to store training and testing MSE, and then write a loop to perform boosting on the training set with 1,000 trees for the pre-defined range of values of the shrinkage parameter, λ. Then, plot the training set and test set MSE across shrinkage values.

```
n.trees <- 1000

MSE <- tibble(train_mse  = vector(mode = "numeric", length = length(lambda)),
              test_mse = vector(mode = "numeric", length = length(lambda)),
              lambda = lambda)

for (i in seq_along(lambda)){
  boost <- gbm(
    biden~.,
    data = train,
    n.trees = n.trees,
    distribution = "gaussian",
    shrinkage = lambda[i],
    ) # default ineraction depth 1

  MSE$train_mse[i] <- predict(boost, newdata = train, n.trees = n.trees) %>%
    {mean((. - train$biden)^2)}
```

```
  MSE$test_mse[i] <- predict(boost, newdata = test, n.trees = n.trees) %>%
    {mean((. - test$biden)^2)}
}

MSE %>%
  ggplot(aes(x = lambda)) +
  geom_line(aes(y = train_mse, color = "Train MSE")) +
  geom_line(aes(y = test_mse, color = "Test MSE")) +
  labs(y = "MSE", x = "Lambda",
       title = "Train & Test MSE of Boosting with 1000 Trees") +
  scale_color_brewer(palette = "Set1")
```



4. (10 points) The test MSE values are insensitive to some precise value of λ as long as its small enough. Update the boosting procedure by setting λ equal to 0.01 (but still over 1000 trees). Report the test MSE and discuss the results. How do they compare?

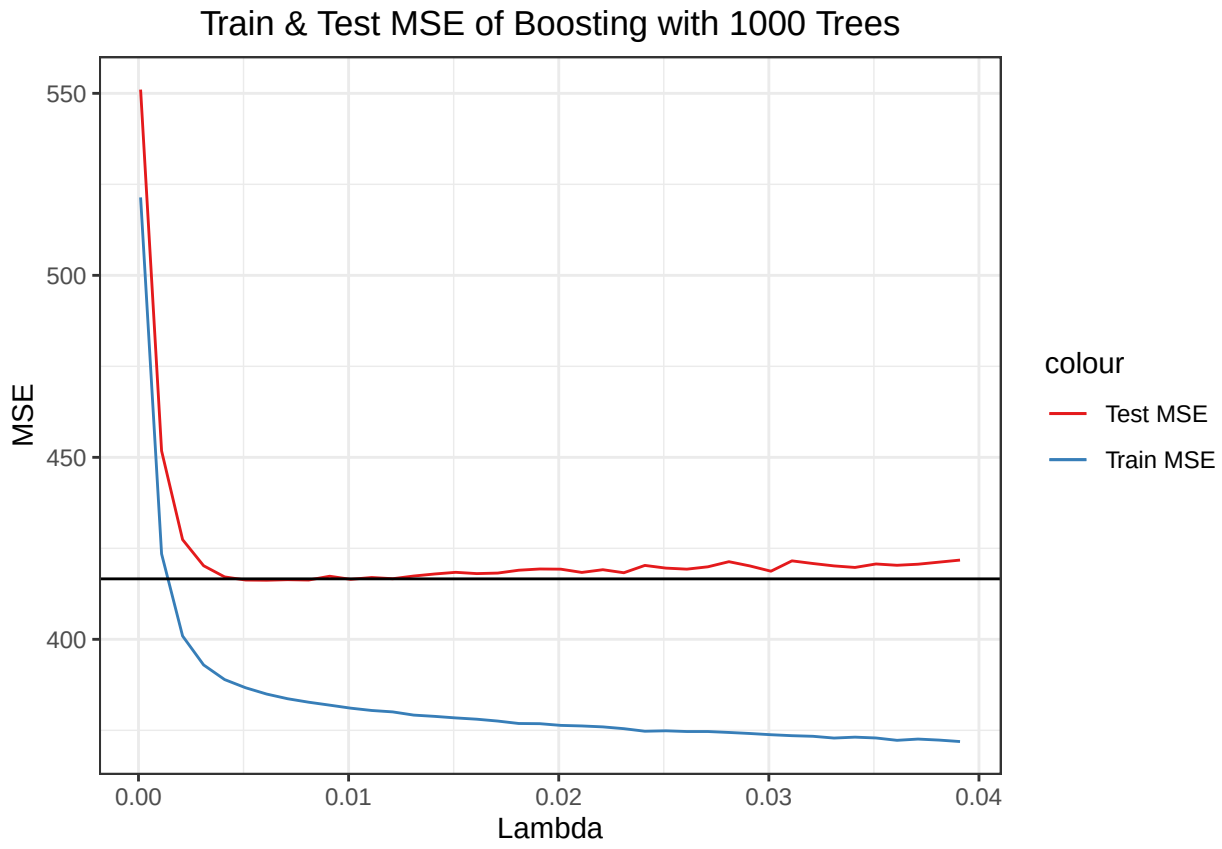
The test MSE of using 0.01 lambda is 416.6126. By adding a horizontal line equal to 416.6126 back to the original plot, we can see that the MSE is relatively low, comparing to the results of other lambda values, and very close to the optimal lambda.

```
boostFit <- gbm(biden~.,
    data = train,
    n.trees = n.trees,
    distribution = "gaussian",
```

```
    shrinkage = 0.01)

boost_mse <- predict(boostFit, newdata = test, n.trees = n.trees) %>%
  {mean((. - test$biden)^2)}

cat("The test MSE of applying boosting with lambda of 0.01 is", boost_mse)
```

```
## The test MSE of applying boosting with lambda of 0.01 is 416.6126
```

```
MSE %>%
  ggplot(aes(x = lambda)) +
  geom_line(aes(y = train_mse, color = "Train MSE")) +
  geom_line(aes(y = test_mse, color = "Test MSE")) +
  labs(y = "MSE", x = "Lambda",
       title = "Train & Test MSE of Boosting with 1000 Trees") +
  geom_hline(yintercept = boost_mse) +
  scale_color_brewer(palette = "Set1")
```



5. (10 points) Now apply bagging to the training set. What is the test set MSE for this approach?

```
bagging <- bagging(
  biden ~ .,
  data = train,
  coob = TRUE,
```

```
)
bagging_mse <- predict(bagging, newdata = test) %>%
  {mean((. - test$biden)^2)}; cat("The test MSE of using bagging is", bagging_mse)
```

```
## The test MSE of using bagging is 415.8213
```

6. (10 points) Now apply random forest to the training set. What is the test set MSE for this approach?

```
rf <- randomForest(biden ~ .,
                   data = train)
rf_mse <- predict(rf, newdata = test) %>%
  {mean((. - test$biden)^2)}; cat("The test MSE of using the random forest approach is", rf_mse)
```

```
## The test MSE of using the random forest approach is 424.9464
```

7. (5 points) Now apply linear regression to the training set. What is the test set MSE for this approach?

```
ols <- lm(biden~., data = train)
ols_mse <- predict(ols, test) %>% {mean((. - test$biden)^2)}; cat("The test MSE of using linear regressi
```

```
## The test MSE of using linear regression is 414.3304
```

8. (5 points) Compare test errors across all fits. Discuss which approach generally fits best and how you concluded this.

According to the table below, we can see that linear regression has the lowest test MSE, while the test MSEs from the four techniques are quite similar. Random Forest has greatest test MSE, possible due to the problem of overfitting. It is important to keep in mind that the results are conditional on this specific train-test split, but in general, linear regression is more consistent while less prone to overfit than nonparametric methods.

```
tibble(Model = c("Boost", "Bagging", "Random Forest", "Linear Regression"),
       MSE = c(boost_mse, bagging_mse, rf_mse, ols_mse)) %>% kable()
```

Model	MSE
Boost	416.6126
Bagging	415.8213
Random Forest	424.9464
Linear Regression	414.3304

Support Vector Machines

1. Create a training set with a random sample of size 800, and a test set containing the remaining observations.

```
set.seed(123)
split <- sample(1: dim(OJ)[1], 800)
```

```
train <- OJ[split, ]
test <- OJ[-split, ]
```

2. (10 points) Fit a support vector classifier to the training data with `cost = 0.01`, with `Purchase` as the response and *all* other features as predictors. Discuss the results.

According to the svm results, 619 support vectors are used, 308 for CH and 311 for MM. The large number of support vector is expected due to the small cost, which leads to a narrow margin and hence less tolerance of violations. If we look at the confusion matrix generated in Q.3, we can see that our model is very prone to type I error (false positive-MM). Almost half of MM are misclassified.

```
svmFit <- svm(Purchase ~ .,
              data = train,
              kernel = "linear",
              cost = 0.01,
              scale = FALSE,
              type = "C-classification"); summary(svmFit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "linear", cost = 0.01,
##     type = "C-classification", scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  619
##
##  ( 308 311 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

3. (5 points) Display the confusion matrix for the classification solution, and also report both the training and test set error rates.

```
cat("The train MSE is",
    predict(svmFit, train) %>% {mean((. != train$Purchase)^2)}, "\n",
    "The test MSE is",
    predict(svmFit, test) %>% {mean((. != test$Purchase)^2)})
```

```
## The train MSE is 0.225
##  The test MSE is 0.237037
```

```r
cat("\n", "Confusion Matrix of Training MSE")
```

```
##
##  Confusion Matrix of Training MSE
```

```r
confusionMatrix(predict(svmFit, train), train$Purchase)$table
```

```
##           Reference
## Prediction  CH  MM
##         CH 429 122
##         MM  58 191
```

```r
cat("\n", "Confusion Matrix of Test MSE")
```

```
##
##  Confusion Matrix of Test MSE
```

```r
confusionMatrix(predict(svmFit, test), test$Purchase)$table
```

```
##           Reference
## Prediction  CH  MM
##         CH 148  46
##         MM  18  58
```

4. (10 points) Find an optimal cost in the range of 0.01 to 1000 (specific range values can vary; there is no set vector of range values you must use)

```r
set.seed(2345)

tune_c <- tune(svm,
               Purchase ~ .,
               data = train,
               kernel = "linear",
               type = "C-classification",
               ranges = list(cost = seq(0.01, 20, by = 0.1)))
```

5. (10 points) Compute the optimal training and test error rates using this new value for cost. Display the confusion matrix for the classification solution, and also report both the training and test set error rates. How do the error rates compare? Discuss the results in substantive terms (e.g., how well did your optimally tuned classifer perform? etc.)

In Q.2-3, we used a cost value of 0.01, while the optimal cost value we found is 3.41. By comparing the Confusion Matrixs and MSEs in Q.5 and Q.3, we can see that the prediction accuracy increased by approximately 0.06~0.07 after relaxing the cost value and allowing for more violations. While the test MSE is slightly greater than the train MSE, the overall performance of our model is quite well. Over 80% of the observations are put in the correct categories. We also note that our mode is less prone to type I error, comparing to previous model.

```
tuned_model <- tune_c$best.model

cat("Optimal cost:", tune_c$best.parameters$cost, "\n",
    "The train MSE is",
    predict(tuned_model, train) %>% {mean((. != train$Purchase)^2)}, "\n",
    "The test MSE is",
    predict(tuned_model, test) %>% {mean((. != test$Purchase)^2)})
```

```
## Optimal cost: 1.11
##  The train MSE is 0.15875
##  The test MSE is 0.1555556
```

```
cat("\n", "Confusion Matrix of Training MSE")
```

```
##
##  Confusion Matrix of Training MSE
```

```
confusionMatrix(predict(tuned_model, train), train$Purchase)$table
```

```
##            Reference
## Prediction  CH  MM
##         CH 428  68
##         MM  59 245
```

```
cat("\n", "Confusion Matrix of Test MSE")
```

```
##
##  Confusion Matrix of Test MSE
```

```
confusionMatrix(predict(tuned_model, test), test$Purchase)$table
```

```
##            Reference
## Prediction  CH  MM
##         CH 148  24
##         MM  18  80
```