HW 3

Michael Carrion

February 17, 2020

Decision Trees

1.

```
set.seed(114)
myE <- read_csv("data/nes2008.csv")
```

```
## Parsed with column specification:
## cols(
##   biden = col_double(),
##   female = col_double(),
##   age = col_double(),
##   educ = col_double(),
##   dem = col_double(),
##   rep = col_double()
## )
```

```
p <- dplyr::select(myE,-c(biden))
myS = seq(from=0.001,to=0.04, by=0.001)
```

2.

```
set.seed(114)
sample_size <- 0.75*nrow(myE)
ind <- sample(seq_len(nrow(myE)),sample_size)
train <- myE[ind,]
test <- myE[-ind,]
```

3.

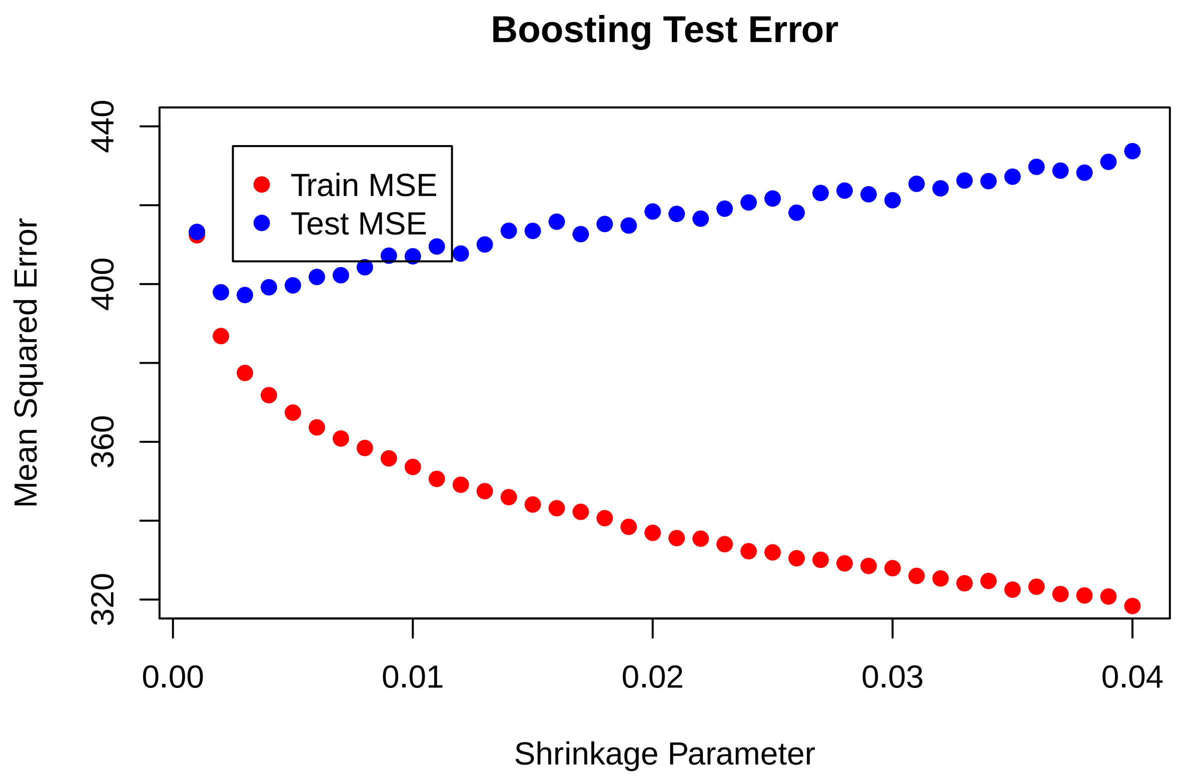
```
myLTest <- list()
myLTrain <- list()

for (val in myS) {
    boost.myE <- gbm(biden ~ .,
                        data=train,
                        distribution="gaussian",
                        n.trees=1000,
                        shrinkage=val,
                        interaction.depth = 4)

    predsTest = predict(boost.myE, newdata=test, n.trees = 1000)
    predsTrain = predict(boost.myE, newdata=train, n.trees = 1000)

    mseTest = mean((predsTest - test$biden)^2)
    mseTrain = mean((predsTrain - train$biden)^2)
    myLTest <- c(myLTest, mseTest)
    myLTrain <- c(myLTrain, mseTrain)
}
```

```r
plot(myS, myLTrain, pch=19, col = "red",
     ylab="Mean Squared Error", xlab="Shrinkage Parameter",
     main="Boosting Test Error", ylim = c(320,440))
points(myS, myLTest, pch = 19, col = "blue")
legend(0.0025,435,legend=c("Train MSE", "Test MSE"),
       col=c("red", "blue"),pch=19)
```



4.

```r
boost.myE <- gbm(biden ~ .,
                        data=train,
                        distribution="gaussian",
                        n.trees=1000,
                        shrinkage=0.01,
                        interaction.depth = 4)

preds= predict(boost.myE, newdata=test, n.trees = 1000)
mseBoost = mean((preds - test$biden)^2)
mseBoost
```

```
## [1] 407.4032
```

```r
min(unlist(myLTest)) #for comparison
```

```
## [1] 397.2273
```

The test set MSE is 407.40 From the figure above, we can see that a shrinkage value of 0.01 is a bit larger than the optimal shrinkage value (the minimum across all shrinkage values was 397.23). A smaller shrinkage value (at about 0.003) would likely be optimal.

5.

```r
#Note: changing nbagg did not significantly change OOB mse, so, we commented it out
#myL <- list()
#bagSize = seq(from=10,to=200, by=10)
#for (i in bagSize) {
    myBag <- bagging(
      formula = biden ~ .,
      data = train,
      nbagg = 100, #i,
      coob = TRUE)
    #myL <- c(myL,myBag$err)
#}
predsBagging <- predict(myBag, newdata = test)
mseBagging = mean((predsBagging - test$biden)^2)
mseBagging
```

```
## [1] 397.9834
```

The bagged model yields a test set MSE of about 397.98, and therefore performs better than the boosted model above with $\lambda = 0.01$, but about the same as the boosted model with the more optimal value of $\lambda = 0.003$.

6.

```r
rf <- randomForest(biden ~ ., data = train)
predsRF <- predict(rf, newdata = test)
mseRF = mean((predsRF - test$biden)^2)
mseRF
```

```
## [1] 402.211
```

The random forest model yields a test set MSE of about 402.21, and thus performs marginally worse than both the optimal boosted and bagged models above.

7.

```r
ols <- lm(biden ~ ., data = train)
predsOls <- predict(ols, newdata = test)
mseOls = mean((predsOls - test$biden)^2)
mseOls
```

```
## [1] 396.6157
```

The linear regression model yields a test set MSE of about 396.62, and thus performs marginally better than all three models above.

8. We get the following table of methods and their corresponding test set MSEs.

```
##            Method      MSE
## 1        Boosting 397.2273
## 2         Bagging 397.9834
## 3   Random Forest 402.2110
## 4 Lin. Regression 396.6157
```

Therefore, we see that the linear regression model yields the lowest test set MSE, and thus is likely the best model we generated. However, the models above performed relatively similarly (that is, after tuning the shrinkage parameter of the boosted model.)

Support Vector Machines

1.

```
set.seed(114)
oj <- OJ
ind <- sample(seq_len(nrow(oj)),800)
train <- oj[ind,]
test <- oj[-ind,]
```

2.

```
set.seed(114)
myClass = svm(formula = Purchase ~ .,
                 data = train,
                 type = 'C-classification',
                 kernel = 'linear',
                 cost = 0.01,
                 scale = FALSE)
summary(myClass)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, type = "C-classification",
##     kernel = "linear", cost = 0.01, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  614
##
##  ( 307 307 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

While it's hard to visualize the classifier because there exist many different features (and thus many dimensions), we see from the summary that the classifier used 614 support vectors, 307 belonging to the Citrus Hill class and 307 belonging to Minute Maid. This large number of support vectors may be the result of our relatively small cost value (0.01). Further, because we use "Purchase"" as the response variable, we attempt to classify each purchase as either Citrus Hill or Minute Maid.

3.

```
set.seed(114)
table(predicted = predict(myClass, test), true = test$Purchase) #test set confusion matrix
```

```
##          true
## predicted  CH  MM
##        CH 147  53
##        MM  14  56
```

```
table(predicted = predict(myClass,train), true = train$Purchase) #training set confusion matrix
```

```
##          true
## predicted  CH  MM
##        CH 453 150
##        MM  39 158
```

Thus, on the test set, we correctly classified $147 + 56 = 203$ of the 270 observations. We incorrectly predicted 53 Minute Maid (MM) purchases and 14 Citrus Hill (CH) purchases. Thus, our test set error rate is $67/270 = 25\%$. Further, on the training set, we correctly classified $453 + 158 = 611$ of the 800 observations. Thus, our training set error rate is $189/800 = 24\%$

4.

```
set.seed(114)
tunedClass <- tune(svm,
               Purchase ~ .,
               data = train,
               kernel = "linear",
               ranges = list(cost = c(0.01, 0.1, 1, 10, 100, 200, 500, 1000)))

summary(tunedClass)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.175
##
## - Detailed performance results:
##    cost   error dispersion
## 1 1e-02 0.17500 0.03173239
## 2 1e-01 0.17500 0.03486083
## 3 1e+00 0.17500 0.02825971
## 4 1e+01 0.17875 0.02503470
## 5 1e+02 0.18000 0.02713137
## 6 2e+02 0.17875 0.02703521
## 7 5e+02 0.17875 0.02703521
## 8 1e+03 0.17875 0.02503470
```

```
tuned_model <- tunedClass$best.model
summary(tuned_model)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train,
##     ranges = list(cost = c(0.01, 0.1, 1, 10, 100, 200, 500, 1000)),
##     kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
```

```
##
## Number of Support Vectors:  350
##
##  ( 174 176 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Thus, the optimal cost value is 0.1 This yields a support vector classifier with 350 support vectors.

5.

```r
set.seed(114)
table(predicted = predict(tuned_model, test), true = test$Purchase) #test set confusion matrix
```

```
##          true
## predicted  CH  MM
##        CH 144  27
##        MM  17  82
```

```r
table(predicted = predict(tuned_model,train), true = train$Purchase) #training set confusion matrix
```

```
##          true
## predicted  CH  MM
##        CH 433  73
##        MM  59 235
```

With the optimal cost value of 0.1, our classification model correctly classified $144 + 82 = 226$ of the 270 test set observations. We incorrectly classified 27 Minute Maid purchases and 17 Citrus Hill purchases, for an error rate of $27 + 17 = 16\%$. Further, with the optimal cost value, our classification model correctly classified $433 + 235 = 668$ of the 800 training set observations. We had an error rate of $59 + 73 = 132/800 = 17\%$. Since these error rates are lower than those of our original model, this new model with a cost of 10 slightly outperforms our original model. Specifically, when using the optimal cost value, we correctly classified 9% more of the test set observations than the original model. Further, in the original model, most of our misclassifications were Minute Maid we incorrectly classified as Citrus Hill; in our revised model, there was a more equal amount of Minute Maid and Citrus Hill Classifications. Thus, this increased cost of margin violations lead to a more highly-fit model (and fewer misclassifications).