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)
```

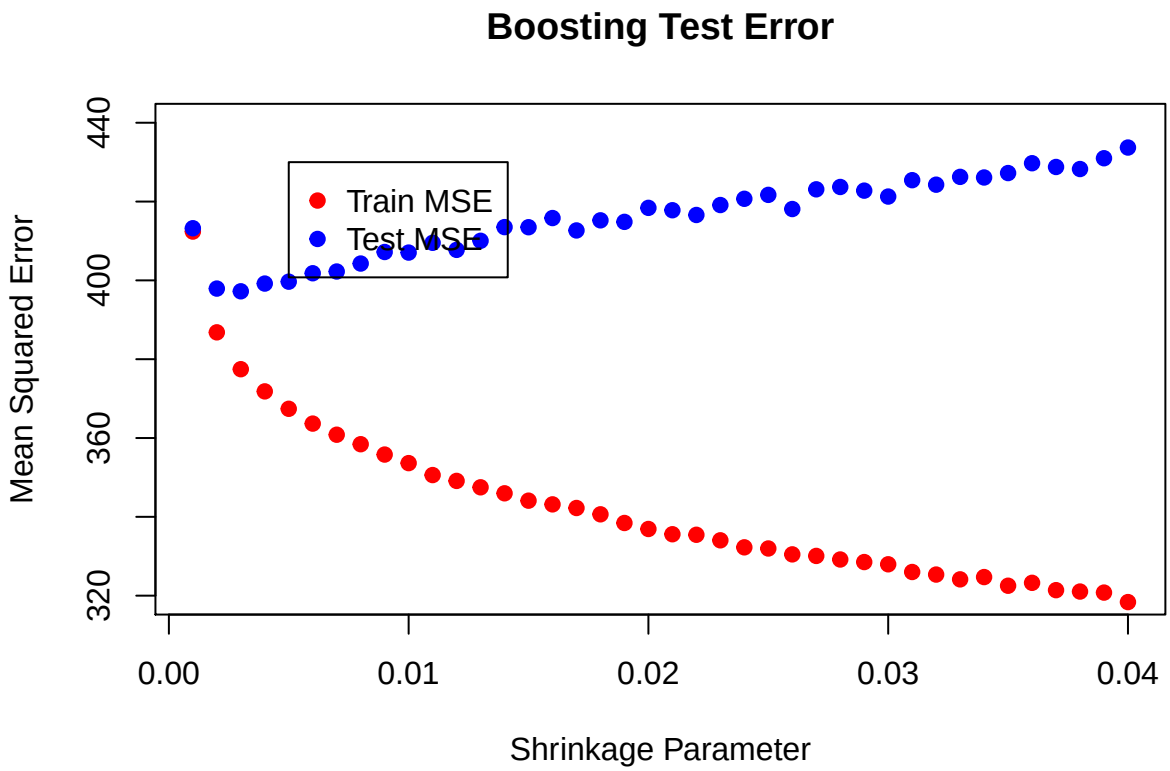
2.

```
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)
}

plot(myS, myLTrain, pch=19, col = "red", ylab="Mean Squared Error", xlab="Shrinkage Parameter", main="B
points(myS, myLTest, pch = 19, col = "blue")
```

```
legend(0.005,430,legend=c("Train MSE", "Test MSE"),
       col=c("red", "blue"),pch=19)
```



4.

```
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
```

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

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

The test set mse is 371.81. Thus, we can see that a shrinkage value of 0.01 yields one of the lowest MSEs for the test results (the minimum across all shrinkage values was 370.93. Therefore $\lambda = 0.01$ is a near-optimal shrinkage value for our model.

5.

```
#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 379, and therefore performs slightly worse than the boosted model above.

6.

```
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 388, and thus performs worse than both the boosted and bagged models above.

7.

```
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 374, and thus performs slightly worse than the boosted model, but slightly better than both the bagged model and the random forest model above.

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

```
##            Method      MSE
## 1        Boosting 407.4032
## 2         Bagging 397.9834
## 3   Random Forest 402.2110
## 4 Lin. Regression 396.6157
```

Therefore, we see that Boosting (with the near-optimal shrinkage parameter of $\lambda = 0.01$) yields the lowest test set MSE, and thus is likely the best model we generated. However, it's important to note that other, less-optimal values of λ we tested, yielding MSEs closer to those of other models. That is, the boosting model had an "unfair advantage" of hyperparameter tuning, so it would be interesting to see if other models perform better than the boosted model when some of their hyperparameters are tuned as well (e.g changing the number of trees in the random forest).

Support Vector Machines

1.

```
oj <- OJ
ind <- sample(seq_len(nrow(oj)),800)
```

```
train <- oj[ind,]
test <- oj[-ind,]
```

2.