Rahul Rangwani
Problem Set 3
February 17, 2020
Intro to Machine Learning

Decision Trees

1. To set up the problem, I ran the following code:

```
dat <- read.csv('nes2008.csv')
set.seed(17)
p <- 5
lambda <- seq(from = .0001, to = .04, by = .001)
```

2. To create my test and training sets, I ran the following code:

```
gen_samples <- sample(seq_len(nrow(dat)), 1355)
train <- dat[gen_samples,]
test <- dat[-gen_samples,]
```

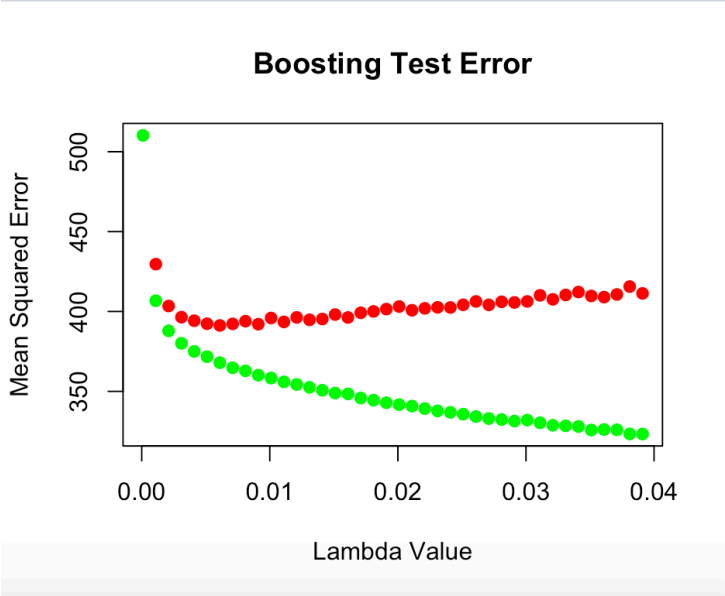
3. For this part, I ran the following code:

```
library(gbm)
trainMSEls <- c()
testMSEls <- c()
for (lamb in lambda) {
  boost.dat <- gbm(biden ~ .,
                   data=train,
                   distribution="gaussian",
                   n.trees=1000,
                   shrinkage=lamb,
                   interaction.depth = 4)

  predtrain = predict(boost.dat, newdata=train, n.trees = 1000)
  predtest = predict(boost.dat, newdata=test, n.trees = 1000)

  msetrain = mean((predtrain - train$biden)^2)
  msetest = mean((predtest - test$biden)^2)

  trainMSEls <- c(trainMSEls, msetrain)
  testMSEls <- c(testMSEls, msetest)

}
```

For the plot, I ran the following code:

```
plot(lambda, trainMSEls,
     pch=19,
     ylab="Mean Squared Error",
     xlab="Lambda Value",
     main="Boosting Test Error",
     col="green")
points(lambda, testMSEls, pch = 19, col = "red")
```

This is the resulting plot:



4. For this part, I ran the following code:

```
boost.dat <- gbm(biden ~ .,
                 data=train,
                 distribution="gaussian",
                 n.trees=1000,
                 shrinkage=.01,
                 interaction.depth = 4)

predtestboost = predict(boost.dat, newdata=test, n.trees = 1000)
msetestboost = mean((predtestboost - test$biden)^2)
msetestboost
summary(testMSEls)
```

For a lambda value of .01, my MSE was 394.083. To compare to the loop, I looked at a summary of the test MSE list, and found that the minimum value was 391.3 and the mean was 406.5. So, it seems as if the .01 lambda value performed better than *most* other specifications with different lambda values.

5. To run a bagging model, I ran the following code:

```
bagtest <- bagging(
  formula = biden ~ .,
  data = train,
  nbagg = 50,
  coob = TRUE)

predtestbag = predict(bagtest, newdata=test)
msetestbag = mean((predtestbag - test$biden)^2)
msetestbag
```

My test MSE here was 402.3302, which is fairly higher than the test MSE of the boosting model, indicating a worse performance.

6. To run a random forest model, I ran the following code:

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

My test MSE here was 414.7502, indicating a significantly worse performance than the boosting and bagging models.

7. To run a linear regression, I ran the following code:

```
lmMod <- lm(biden ~ .,
            data = train)
predtest = predict(lmMod, newdata=test)
msetestols = mean((predtest - test$biden)^2)
msetestols
```

My test MSE here was 393.8684, the best performing model of the four.

8. The test MSE for boosting was 394.083, for bagging was 402.3302, for random forest was 414.7502, and for linear regression was 393.8684. As we can see, the linear model performed the best. However, the boosting model, after tuning the hyperparameter of shrinkage factor, performed almost as well. Bagging was fairly close in performance, and random forest was significantly worse.

Support Vector Machines

1. To split my data, I ran the following code:

```
datOJ <- OJ
init_split <- initial_split(data = datOJ, prop = .747663)
OJ_train <- training(init_split)
OJ_test <- testing(init_split)
```

2. To get my model, I ran the following code:

```
svmfit <- svm(Purchase ~ .,
              data = OJ_train,
              kernel = "linear",
              cost = .01,
              scale = FALSE); summary(svmfit)
```

The results were as follows:

```
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01

Number of Support Vectors:  627

 ( 314 313 )


Number of Classes:  2

Levels:
 CH MM
```

Here, we see that our SVM has 627 total support vectors for our two classes – 314 vectors for Citrus Hill and 313 vectors for Minute Maid.

3. For the confusion matrices, I ran the following code:

```
table(predicted = predict(svmfit,OJ_train), true = OJ_train$Purchase)

table(predicted = predict(svmfit, OJ_test), true = OJ_test$Purchase)
```

(Training)	true	
predicted	CH	MM
CH	460	178
MM	27	135

(Test)	true	
predicted	CH	MM
CH	160	58
MM	6	46

For the training set, we see that we have 460 correct Citrus Hill predictions, and 135 correct Minute Maid predictions. We have 178 incorrect Citrus Hill predictions and 27 incorrect Minute Maid predictions, resulting in an error rate of (27+178)/800 = 25.6%. For the test set, we see that we have 160 correct Citrus Hill predictions, and 46 correct Minute Maid predictions. We have 58 incorrect Citrus Hill predictions and 6 incorrect Minute Maid predictions, resulting in an error rate of (6+58)/270 = 23.7%.

4. To tune the model, I ran the following code:

```
set.seed(1717)

tune_OJ <- tune(svm,
                Purchase ~ .,
                data = OJ_train,
                kernel = "linear",
                ranges = list(cost = c(0.01, 0.1, 1, 5, 10, 100, 250, 500, 750, 1000)))


summary(tune_OJ)

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

This was the output:

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
    5

- best performance: 0.16375

- Detailed performance results:
      cost   error dispersion
1     0.01 0.17750 0.05458174
2     0.10 0.16750 0.05565269
3     1.00 0.16875 0.05179085
4     5.00 0.16375 0.04980866
5    10.00 0.16500 0.05096295
6   100.00 0.16875 0.05145454
7   250.00 0.16750 0.05041494
8   500.00 0.16750 0.05041494
9   750.00 0.16375 0.05118390
10 1000.00 0.16750 0.05041494
```

As we can see, the model with the best performance (lowest error) was with cost 5.

5. To get the confusion matrices, I ran the following code:

```
table(predicted = predict(tuned_model,OJ_train), true = OJ_train$Purchase)

table(predicted = predict(tuned_model, OJ_test), true = OJ_test$Purchase)
```

```
              true                    true
predicted   CH  MM  predicted   CH  MM
       CH  428  71         CH  146  25
       MM   59 242         MM   20  79
     (training)              (test)
```

For the training set, we see that we have 428 correct Citrus Hill predictions, and 242 correct Minute Maid predictions. We have 71 incorrect Citrus Hill predictions and 59 incorrect Minute Maid predictions, resulting in an error rate of (71+59)/800 = 16.3%. For the test set, we see that we have 146 correct Citrus Hill predictions, and 79 correct Minute Maid predictions. We have 25 incorrect Citrus Hill predictions and 20 incorrect Minute Maid predictions, resulting in an error rate of (20+25)/270 = 16.7%. The tuned model preforms significantly better than the previous model, which had error rates of 25.6% and 23.7% for training and testing sets, respectively. If we look at the breakdown

of the misclassifications in the original model, we see that they were concentrated around incorrectly classifying as Minute Maid. With the new model, the misclassifications were more balanced between the two brands.