Problem Set 3

Siva Bhavani

2/11/2020

Problem Set 3: Trees & Machines

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

```
rm(list=ls())
nes2008 <- read_csv("C:/Users/sbhavani/Documents/problem-set-3/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()
## )
```

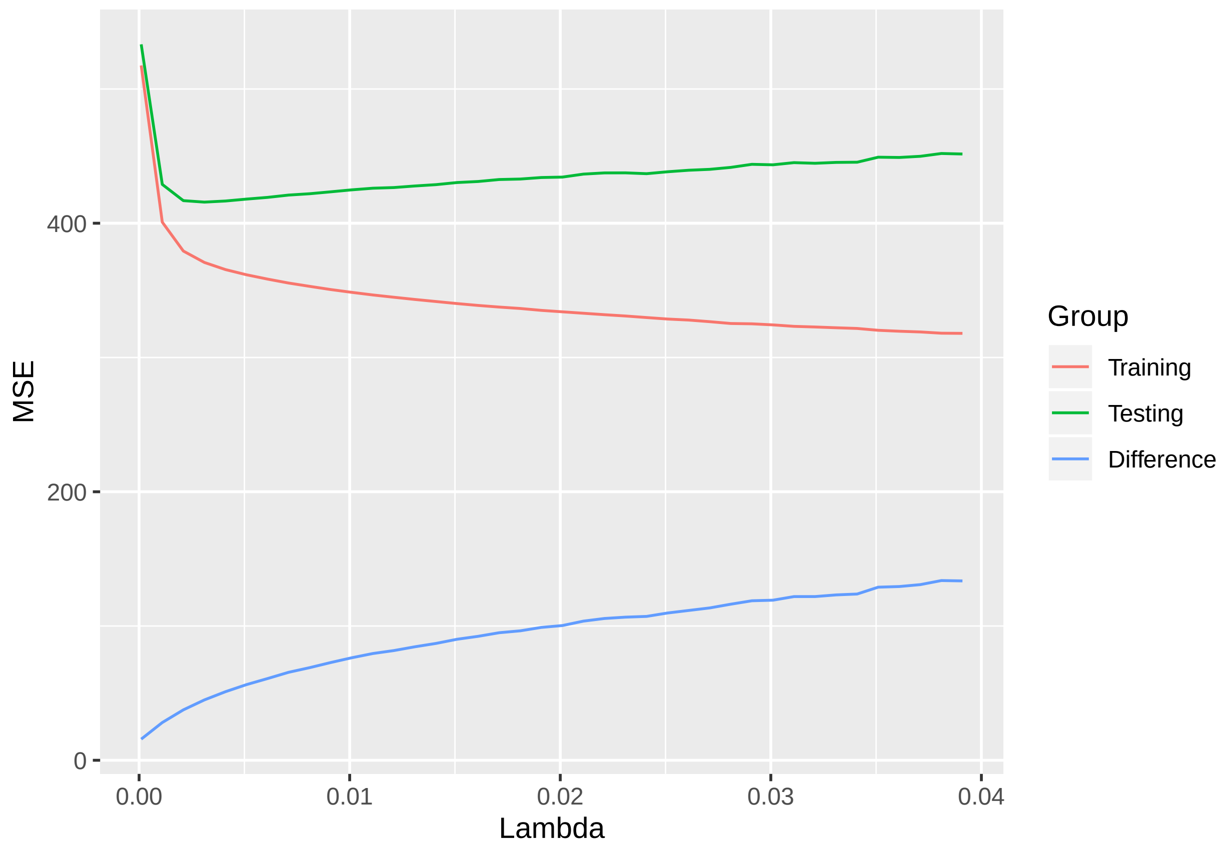
```
set.seed(122)
p = ncol(nes2008)-1
lambda = seq(0.0001, 0.04, 0.001)
```

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.

```
sample_size = floor(0.75*nrow(nes2008))
picked = sample(seq_len(nrow(nes2008)),size = sample_size)
training =nes2008[picked,]
testing =nes2008[-picked,]
```

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.

```r
lambda2 = rep(lambda, times=2)
group = rep(c(1,2), each=40)

MSE = as.data.frame(matrix(nrow = 80, ncol = 3))
names = cbind("Lambda", "MSE", "Group")
colnames(MSE) = names
MSE$Lambda = lambda2
MSE$Group = group


for (x in 1:40) {
set.seed(122)
boost <- gbm(biden ~ .,
                    data=training,
                    distribution="gaussian",
                    n.trees=1000,
                    shrinkage=lambda2[x],
                    interaction.depth = 4)
prediction=predict(boost,newdata=training,n.trees=1000)
MSE$MSE[x] = mean((prediction-training$biden)^2)
}

for (x in 41:80) {
  set.seed(122)
boost <- gbm(biden ~ .,
                    data=training,
                    distribution="gaussian",
                    n.trees=1000,
                    shrinkage=lambda2[x],
                    interaction.depth = 4)
prediction=predict(boost,newdata=testing,n.trees=1000)
MSE$MSE[x] = mean((prediction-testing$biden)^2)
}

MSE$Group = as.factor(MSE$Group)
levels(MSE$Group) = c("Training", "Testing")

difference_in_mse = as.data.frame(matrix(nrow = 40, ncol = 3))
names = cbind("Lambda", "MSE", "Group")
colnames(difference_in_mse) = names
difference_in_mse$Lambda = lambda
difference_in_mse$Group = "Difference"

for (x in 1:40) {
difference_in_mse$MSE[x] = MSE$MSE[x+40] - MSE$MSE[x]
}

MSE = rbind(MSE, difference_in_mse)

ggplot(data=MSE, aes(x=Lambda, y=MSE, group=Group)) +
  geom_line(aes(color=Group))
```



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?

```r
test_MSE = as.data.frame(matrix(nrow = 4, ncol = 2))
names = cbind("Model", "MSE")
colnames(test_MSE) = names
names = cbind("Boosting", "Bagging", "RandomForest", "LinearRegression")
test_MSE$Model[1:4] = names




set.seed(122)
boost <- gbm(biden ~ .,
                    data=training,
                    distribution="gaussian",
                    n.trees=1000,
                    shrinkage=0.01,
                    interaction.depth = 4)
prediction=predict(boost,newdata=testing,n.trees=1000)
test_MSE$MSE[1] =  mean((prediction-testing$biden)^2)
test_MSE$MSE[1]
```

```
## [1] 424.9417
```

The test MSE is 425 while the training MSE is 349. There is a large difference between the two, and with this and higher shrinkage numbers the two MSE's diverge, resulting in a model that is very well fitted to the training data but not as generalizable to the testing data. As can be seen in the figure, the difference between the MSE starts to increase in the testing data set with increasing lambda.

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

```r
set.seed(122)
bag <- bagging(
  formula = biden ~ .,
  data = training,
  nbagg = 100,
  coob = TRUE,
  control = rpart.control(minsplit = 2, cp = 0)
)
prediction=predict(bag,newdata=testing,n.trees=1000)
test_MSE$MSE[2] =  mean((prediction-testing$biden)^2)
test_MSE$MSE[2]
```

```
## [1] 527.6433
```

The test set MSE was 528.

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

```r
set.seed(122)
rf <- randomForest(
  formula = biden ~ .,
  data = training,
  ntree = 1000,
  mtry = 3
)

prediction=predict(rf,newdata=testing,n.trees=1000)
test_MSE$MSE[3] =  mean((prediction-testing$biden)^2)
test_MSE$MSE[3]
```

```
## [1] 453.3018
```

The test set MSE was 453.

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

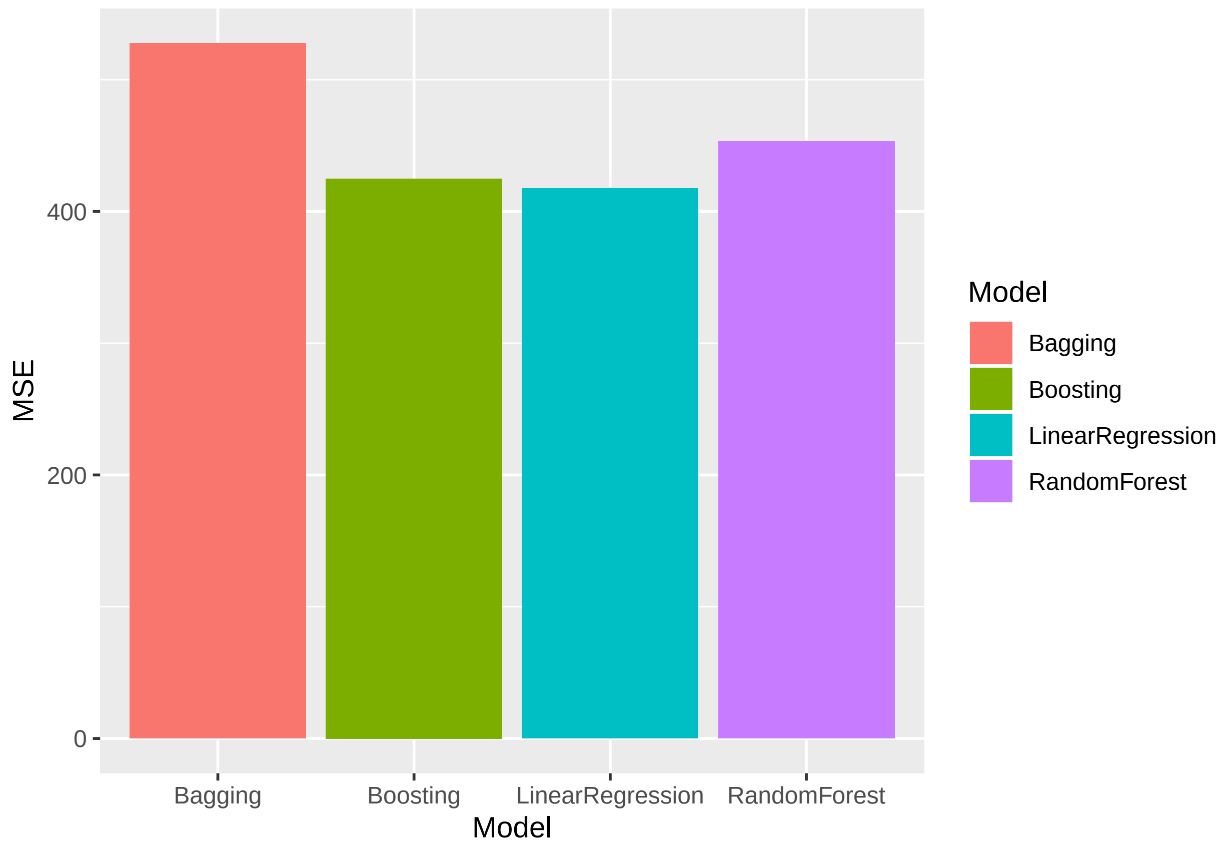
```r
set.seed(122)
lr <- lm(
  formula = biden ~ .,
  data = training
)

prediction=predict(lr,newdata=testing)
test_MSE$MSE[4] =  mean((prediction-testing$biden)^2)
test_MSE$MSE[4]
```

```
## [1] 417.4143
```

The test set MSE was 417.

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

```r
ggplot(data=test_MSE, aes(x=Model, y=MSE)) +
  geom_bar(stat = "identity", aes(fill=Model))
```



Linear regression had the best fit on test data set, with the lowest mean squared error. Boosting followed, while bagging performed the worse.

Support Vector Machines

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

```r
rm(list=ls())
set.seed(122)
OJ <- OJ
sample_size = 800
picked = sample(seq_len(nrow(OJ)),size = sample_size)
training =OJ[picked,]
testing =OJ[-picked,]
```

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.

```r
Grid <-  expand.grid(C = c(0.01))

set.seed(122)
svmModel <- train(
  Purchase ~ .,
  data = training,
  method = "svmLinear",
  tuneGrid  = Grid)

svmModel
```

```
## Support Vector Machines with Linear Kernel
##
## 800 samples
##  17 predictor
##   2 classes: 'CH', 'MM'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 800, 800, 800, 800, 800, 800, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8215237  0.6241575
##
## Tuning parameter 'C' was held constant at a value of 0.01
```

The model had an accuracy of 0.82 and a kappa of 0.61. So there was good performance in prediciting the type of OJ purchased.

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

```r
prediction_train = predict(svmModel, newdata = training)
train_stats = confusionMatrix(prediction_train, training$Purchase)
train_stats$table
```

```
##           Reference
## Prediction  CH  MM
##         CH 422  77
##         MM  58 243
```

```r
error = 1 - train_stats$overall[1]
error
```

```
## Accuracy
##  0.16875
```

```r
prediction_test = predict(svmModel, newdata = testing)
test_stats = confusionMatrix(prediction_test, testing$Purchase)
test_stats$table
```

```
##           Reference
## Prediction  CH  MM
##         CH 155  26
##         MM  18  71
```

```r
error = 1 - test_stats$overall[1]
error
```

```
## Accuracy
## 0.162963
```

The error rate in the training set was 0.17, and the error rate in the testing set was 0.15.

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(122)

Grid <-  expand.grid(C = c(0.01, 0.1, 1, 10, 100, 500, 1000))

svmModel <- train(
  Purchase ~ .,
  data = training,
  method = "svmLinear",
  tuneGrid= Grid)

svmModel
```

```
## Support Vector Machines with Linear Kernel
##
## 800 samples
##  17 predictor
##   2 classes: 'CH', 'MM'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 800, 800, 800, 800, 800, 800, ...
## Resampling results across tuning parameters:
##
##   C      Accuracy   Kappa
##   1e-02  0.8215237  0.6241575
##   1e-01  0.8206499  0.6221501
##   1e+00  0.8198767  0.6205096
##   1e+01  0.8227401  0.6269002
##   1e+02  0.8190541  0.6190864
##   5e+02  0.8183854  0.6178798
##   1e+03  0.8185259  0.6181420
```

```
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 10.
```

Cost of 10 had the highest accuracy, but accuracy was not significantly different across the range of cost values.

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.)

```r
set.seed(122)

Grid <-  expand.grid(C = c(10))

svmModel <- train(
  Purchase ~ .,
  data = training,
  method = "svmLinear",
  tuneGrid= Grid)


prediction_train = predict(svmModel, newdata = training)
train_stats = confusionMatrix(prediction_train, training$Purchase)
train_stats$table
```

```
##           Reference
## Prediction  CH  MM
##         CH 421  75
##         MM  59 245
```

```r
error = 1 - train_stats$overall[1]
error
```

```
## Accuracy
##   0.1675
```

```r
prediction_test = predict(svmModel, newdata = testing)
test_stats = confusionMatrix(prediction_test, testing$Purchase)
test_stats$table
```

```
##           Reference
## Prediction  CH  MM
##         CH 152  24
##         MM  21  73
```

```r
error = 1 - test_stats$overall[1]
error
```

```
##  Accuracy
## 0.1666667
```

The error rate for the training data set is 0.17, and the error rate for test data set is 0.17. There is no difference in error between the training and test data sets. As the cost function approaches zero, the partition of the groups would be more similar to a hard margin classifier and wouldn't do well with outliers. However, as the cost function increases towards infinity, the partition of the groups would be much softer and there would be more overlap across the hyperplane. When the model is more sensitive to outliers (as cost approaches zero), it may not perform as well in a different data set (the test dataset). As the cost function approaches infinity, there is going to be more bias but less variance. The current error rate in the test data set is actually equivalent to the training set, so the model seems generalizable and not overfit.