Problem Set 3: Trees & Machines

Jinglin Yang

Feb 2020

```
library(gbm)
library(tidyverse)
library(ISLR)
library(ggplot2)
library(data.table)
library(e1071)
library(randomForest)
```

Decision Trees

Question 1

Set up the data and store some things for later use:

- Set seed

```
> set.seed(1)
```

- Load the data

```
> nes_2008<- read.csv("/Users/jinglin0828/Downloads/nes2008.csv",header = T)
> attach(nes_2008)
```

- Store the total number of features minus the biden feelings in obejct p (p = 5)

```
> p<- ncol(nes_2008)-1
```

- Set λ (shrinkage/learning rate) range from 0.0001 to 0.04,by 0.001

```
> Lambda <- seq(from=0.0001, to=0.04, by=0.001)
```

Question 2

Create a training set consisting of 75% of the observations, and a test set with all remaining obs.

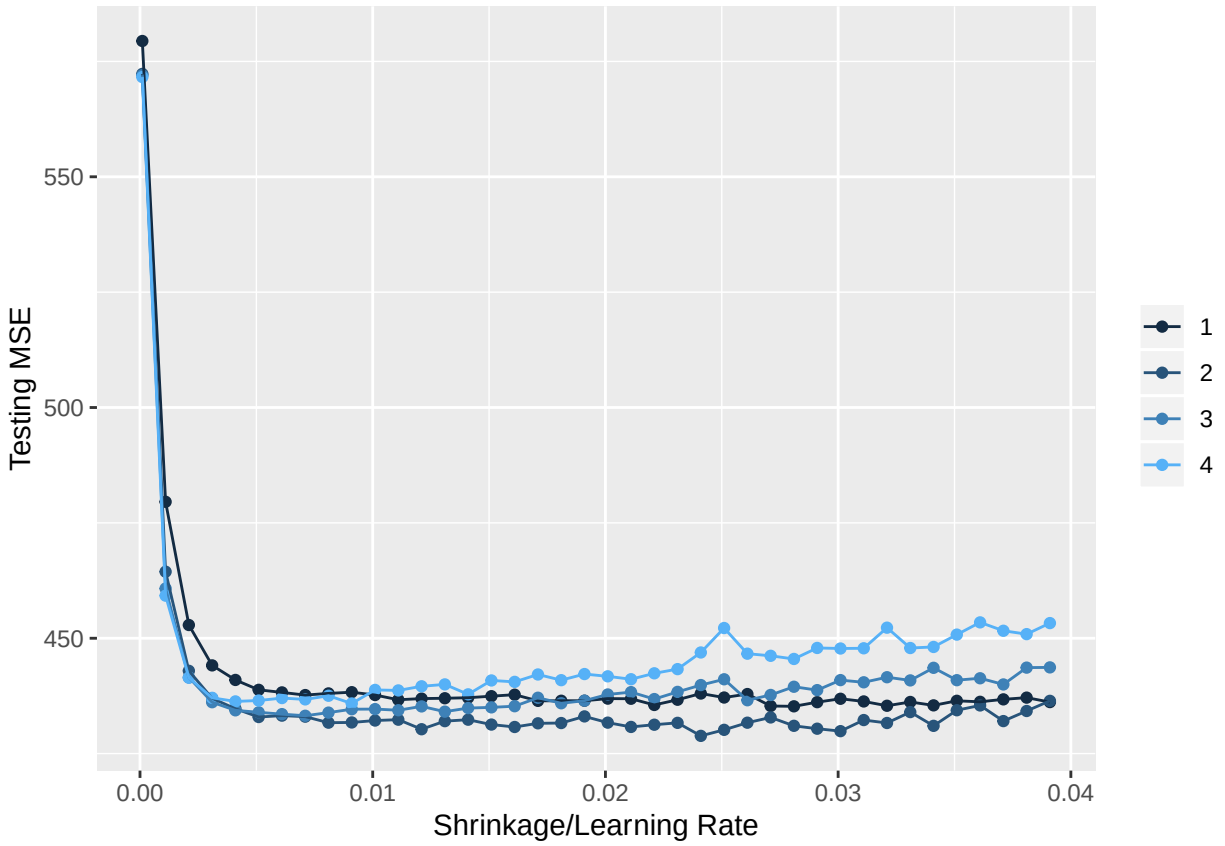
```
> train <- sample(1:nrow(nes_2008),nrow(nes_2008)*0.75)
> nes_train <- nes_2008[train,]
> nes_test <- nes_2008[-train,]
```

Question 3

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 value.

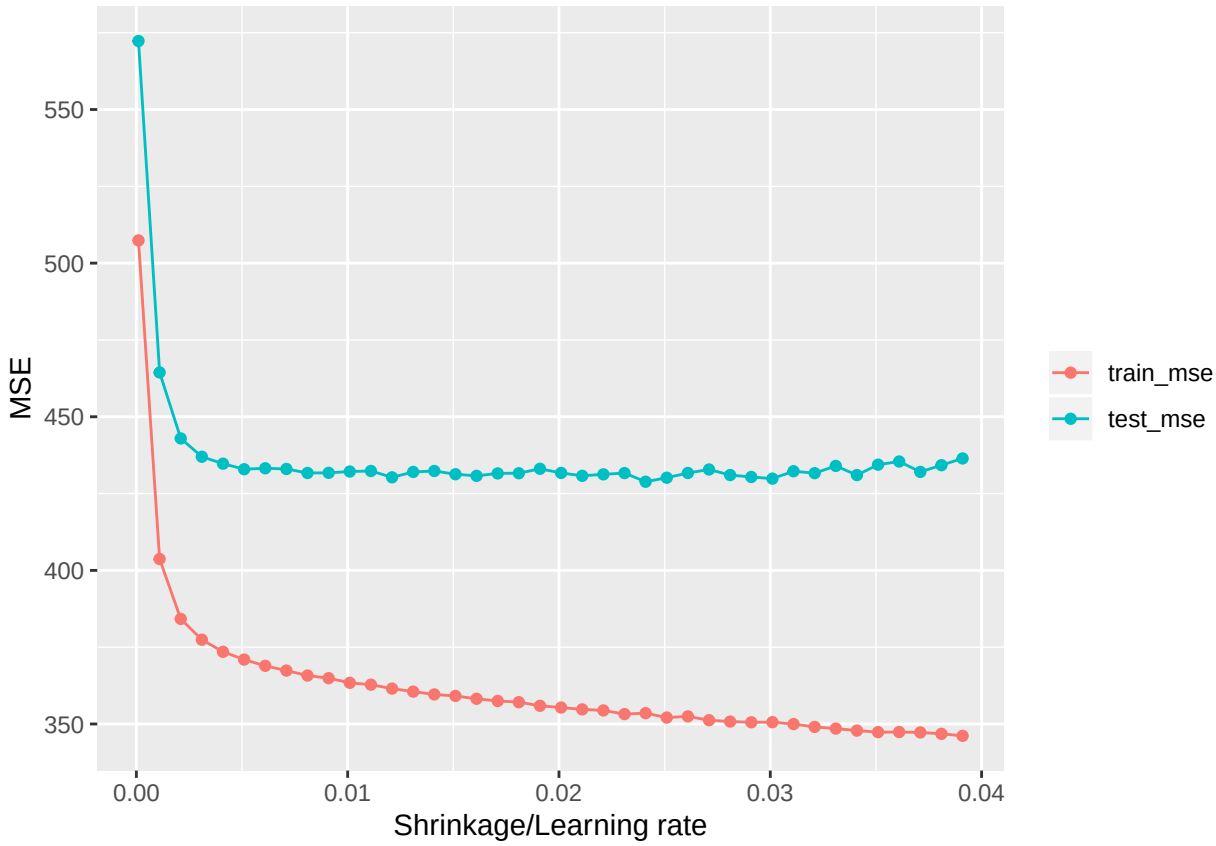
First, I need to decide the optimal interaction depth. Therefore, I try to plot the test set MSE across shrinkage value for different depth.

```
> for (d in 1:4){
+   # create a empty dataframe to store training and testing MSE
+   MSEs <- data.frame(lambda = numeric(0), train_mse = numeric(0), test_mse = numeric(0))
+   # write a loop to perform boosting on the training set
+   for (lam in Lambda){
+    boost.nes <- gbm(biden~.,
+                     data = nes_train,
+                     distribution = "gaussian",
+                     n.trees = 1000,
+                     shrinkage = lam,
+                     interaction.depth = d)
+    # training MSE
+    train_boost_preds <- predict(boost.nes, newdata=nes_train,
+                    n.trees = 1000)
+    train_boost_mse <- mean((train_boost_preds-nes_train$biden)^2)
+
+    # testing MSE
+    test_boost_preds <- predict(boost.nes, newdata=nes_test,
+                    n.trees = 1000)
+    test_boost_mse <- mean((test_boost_preds-nes_test$biden)^2)
+
+    # restore
+    new <- data.frame(lambda = lam, train_mse = train_boost_mse, test_mse = test_boost_mse)
+    MSEs <- rbind(MSEs,new)
+   }
+    d_MSEs <- cbind(d, MSEs)
+    assign(paste0("MSEs",d),d_MSEs)
+ }
>
> # plot the test set MSE across shrinkage value for different d values
> d_MSEs <- rbind(MSEs1,MSEs2,MSEs3,MSEs4)
> ggplot(data = d_MSEs, aes(x = lambda, y = test_mse, group = d, color = d)) +
+   geom_point() +
+   geom_line() +
+   xlab ("Shrinkage/Learning Rate") +
+   ylab ("Testing MSE") +
+   guides(color = guide_legend (title=NULL))
```



From the graph above, we can see that the boosting model with an interaction depth of two perform the best, because it has the samllest testing MSEs across different shrinkage values. Therefore, I choose $interaction.depth = 2$.

```
> # the dataset containing the training set and test set MSE with d = 2
> MSEs <- MSEs2
>
> # plot the training set and test set MSE across shrinkage value
> # reshape the dataset for plotting
> MSE_plot <- melt(MSEs, measure.vars=c("train_mse","test_mse"))
>
> ggplot(data = MSE_plot, aes(x = lambda, y = value, color = variable)) +
+   geom_point() +
+   geom_line() +
+   xlab ("Shrinkage/Learning rate") +
+   ylab ("MSE") +
+   guides(color = guide_legend (title=NULL))
```



Question 4

The test MSE values are insensitive to some precise value of λ as long as it is small enough. Update the boosting procedure by setting λ equal to 0.01 (but still over 1,000 trees). Report the test MSE and discuss the results. How do you compare?

```
> boost.nes1 <- gbm(biden~.,
+                     data = nes_train,
+                     distribution = "gaussian",
+                     n.trees = 1000,
+                     shrinkage = 0.01,
+                     interaction.depth = 1)
> test_boost_preds1 <- predict(boost.nes1, newdata=nes_test,
+                    n.trees = 1000)
> test_boost_mse1 <- mean((test_boost_preds1-nes_test$biden)^2); test_boost_mse1
[1] 438.1969
```

The test MSE for boosting with $\lambda = 0.01$ is 438.1969. From the graph of the training set and test set MSE across λ above, we can see that the tset MSE values are insentitive when λ is greater than 0.01. And the testing MSE with $\lambda = 0.01$ is almost the lowest MSE across different shrinkage λ.

Question 5

Now apply bagging to the training set. What is the test set MSE for this approach?

```
> bagging.nes <- randomForest(biden~.,
+                             data = nes_train,
+                             mtry = p)
> test_bag_preds <- predict(bagging.nes, newdata=nes_test)
> test_bag_mse <- mean((test_bag_preds-nes_test$biden)^2); test_bag_mse
[1] 525.4822
```

The test set MSE for bagging is 525.4822.

Question 6

Now apply random forest to the training set. What is the test set MSE for this approach?

```
> rf.nes <- randomForest(biden~.,
+                             data = nes_train,
+                             ntree = 1000,
+                             mtry = sqrt(p))
> test_rf_preds <- predict(rf.nes, newdata=nes_test,
+                     n.trees = 1000)
> test_rf_mse <- mean((test_rf_preds-nes_test$biden)^2); test_rf_mse
[1] 442.7725
```

The test set MSE for random forest is 442.7725.

Question 7

Now apply linear regression to the training set. What is the test set MSE for this approach?

```
> lm.nes <- lm(biden~., data = nes_train)
> test_lm_preds <- predict(lm.nes, newdata=nes_test)
> test_lm_mse <- mean((test_lm_preds-nes_test$biden)^2); test_lm_mse
[1] 437.1178
```

The test set MSE for linear regression is 437.1178.

Question 8

Compare test errors across all fits. Discuss which approach generally fits best and how you concluded this.

Approach	Testing MSE
Boosting ($\lambda = 0.01$)	438.1969
Bagging	525.4822
Random Forest	442.7725
Linear Regression	437.1178

From the table above, we can see that linear regression fits best, because its testing MSE is the smallest. Also, boosting and random forest have similar performance in this case because their testing MSE is similar to that of linear regression.

As we can see, bagging performs the worst in this case. This is probably because the trees in bagging are actually correlated. And averaging many correlated trees will not substantially reduce variance. But random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees.

Although linear model is much less flexible than the tree models because we restrict a linear relationship, in this case, it performs well. This is probably the true relationship is linear. Generally, linear regressions usually have high bias with low variance. However, tree models usually have low bias and large variance. We need to carefully deal with the variance-bias tradeoff in every case.

Support Vector Machines

Question 1

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

```
> set.seed(100)
> train <- sample(1:nrow(OJ),800)
> oj_train <- OJ[train,]
> oj_test <- OJ[-train,]
```

Question 2

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.

```
> svmfit <- svm(Purchase ~ .,
+               data = oj_train,
+               kernel = "linear",
+               cost = 0.01,
+               scale = FALSE)
> summary(svmfit)

Call:
svm(formula = Purchase ~ ., data = oj_train, kernel = "linear",
    cost = 0.01, scale = FALSE)


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

Number of Support Vectors:  623

 ( 312 311 )
```

```
Number of Classes:  2

Levels:
 CH MM
```

For a linear kernel used with cost = 0.01, there were 623 support vectors, 312 in one class and 311 in the other.

Question 3

Display the confusion matrix for the classification solution, and also report both the training and test set error rates.

```
> # confusion matrix for the training data
> train_pred <- predict(svmfit,oj_train)
> table(train_pred,oj_train$Purchase)

train_pred  CH  MM
        CH 466 177
        MM  22 135
>
> # confusion matrix for the testing data
> test_pred <- predict(svmfit,oj_test)
> table(test_pred,oj_test$Purchase)

test_pred  CH  MM
       CH 157  63
       MM   8  42
>
> # training and test set error rates
> train_error <- 1-mean(train_pred==oj_train$Purchase); train_error
[1] 0.24875
> test_error <- 1-mean(test_pred==oj_test$Purchase); test_error
[1] 0.262963
```

The training set error rate is 0.24875, and the testing set error rate is 0.262963.

Question 4

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)

```
> # use CV to find the optimal cost
> set.seed(100)
> tune_c <- tune(svm,
+                Purchase ~ .,
+                data = oj_train,
+                kernel = "linear",
+                ranges = list(cost = c(0.01,0.1,1,10,100,1000)))
```

```
> summary(tune_c)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.1

- best performance: 0.16875

- Detailed performance results:
   cost   error dispersion
1 1e-02 0.17500 0.04639804
2 1e-01 0.16875 0.03875224
3 1e+00 0.17375 0.04101575
4 1e+01 0.17000 0.03782269
5 1e+02 0.17125 0.04332131
6 1e+03 0.17250 0.04199868
> tuned_model <- tune_c$best.model
> summary(tuned_model)

Call:
best.tune(method = svm, train.x = Purchase ~ ., data = oj_train,
    ranges = list(cost = c(0.01, 0.1, 1, 10, 100, 1000)), kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.1

Number of Support Vectors:  343

 ( 172 171 )


Number of Classes:  2

Levels:
 CH MM
```

From the result, we can see the optimal cost is 0.1.

Question 5

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 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.)

```
> # confusion matrix and error rate for the training data using cost = 1,000
> train_pred_1 <- predict(tuned_model,oj_train)
> table(train_pred_1,oj_train$Purchase)

train_pred_1  CH  MM
          CH 427  63
          MM  61 249
> train_error_1 <- 1-mean(train_pred_1==oj_train$Purchase); train_error_1
[1] 0.155
>
> # confusion matrix for the testing data using cost = 1,000
> test_pred_1 <- predict(tuned_model,oj_test)
> table(test_pred_1,oj_test$Purchase)

test_pred_1  CH  MM
         CH 140  27
         MM  25  78
> test_error_1 <- 1-mean(test_pred_1==oj_test$Purchase); test_error_1
[1] 0.1925926
```

With the optimal cost 0.1, the training set error rate is 0.155, and the testing set error rate is 0.1925926. Both error rates are smaller than those of SVM with cost = 0.01. This implies that the optimally tuned classifer performs better. For a low cost, you aim for a smooth decision surface and for a higher cost, you aim to classify more points correctly. It is also simply referred to as the cost of misclassification. Therefore, there exists a trade off. And we should always tune our classifer to find the optimal model.