# Problem Set 3: Trees & Machines

*Casey Mallon*

## 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 $\lambda$ (shrinkage/learning rate) range from 0.0001 to 0.04, by 0.001

```
set.seed(888)
Biden.data<- read.csv("/Users/cemallon/Documents/Machine_Learning/problem-set-3/data/nes2008.csv")

p<- 5

lambda<- seq(0.0001, 0.04, by=  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 $\lambda$ 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.

```
train=sample(1:nrow(Biden.data), 1355.25)
Biden_training_set<- Biden.data[train,]
Biden_test_set<- Biden.data[-train,]
```

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, $\lambda$. Then, plot the training set and test set MSE across shrinkage values.

```
##Training
boosted_tree_training <- function(lambda){
model <- gbm(biden ~ .,
data = Biden_training_set,
distribution="gaussian",
n.trees=1000,
shrinkage = lambda)
preds <- predict(model, newdata=Biden_training_set, n.trees = 1000)
mse <- mean((Biden_training_set$biden-preds)^2)
mse
}

boost_training_list <- map_dbl(lambda, boosted_tree_training)

boost_training_mse <- as.data.frame(cbind(lambda, boost_training_list)) %>%
rename(mse = `boost_training_list`)
boost_training_mse
```

```
##    lambda      mse
## 1  0.0001 525.2334
## 2  0.0011 434.9060
## 3  0.0021 414.0340
## 4  0.0031 406.4523
## 5  0.0041 403.1265
## 6  0.0051 400.7550
## 7  0.0061 399.0891
## 8  0.0071 397.8075
## 9  0.0081 396.9511
## 10 0.0091 395.9993
## 11 0.0101 395.2217
## 12 0.0111 394.4921
## 13 0.0121 393.8737
## 14 0.0131 393.3442
## 15 0.0141 392.7868
## 16 0.0151 392.5355
## 17 0.0161 391.6955
## 18 0.0171 391.3212
## 19 0.0181 390.9239
## 20 0.0191 390.3740
## 21 0.0201 389.9107
## 22 0.0211 389.9854
## 23 0.0221 389.4931
## 24 0.0231 389.0256
## 25 0.0241 388.7991
## 26 0.0251 388.4756
## 27 0.0261 388.0390
## 28 0.0271 387.6336
## 29 0.0281 387.2530
## 30 0.0291 386.7526
## 31 0.0301 386.6005
## 32 0.0311 386.4890
## 33 0.0321 386.1195
## 34 0.0331 386.0252
## 35 0.0341 385.4788
## 36 0.0351 385.4935
## 37 0.0361 385.1763
## 38 0.0371 385.4318
## 39 0.0381 384.8609
## 40 0.0391 384.6980
```

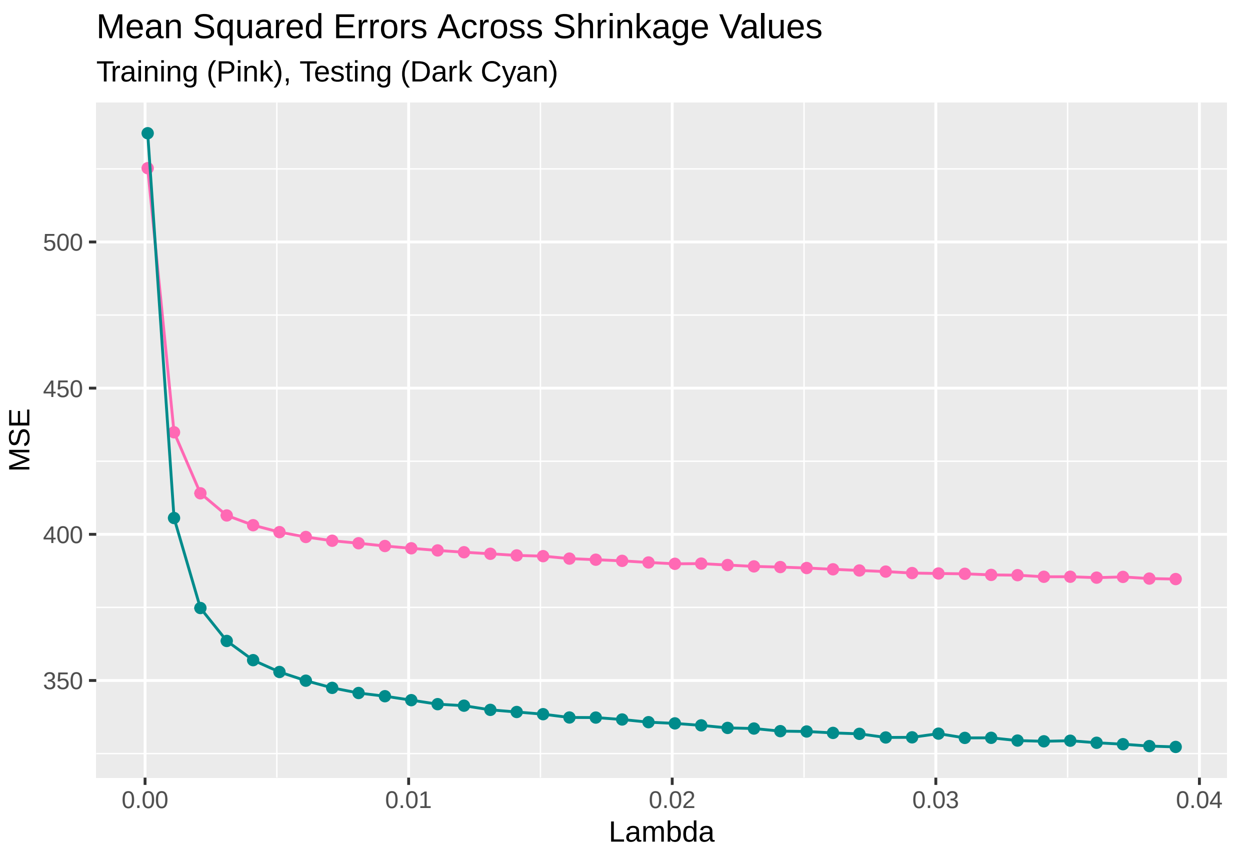
```r
##Testing
boosted_tree_testing <- function(lambda){
model <- gbm(biden ~ .,
data = Biden_test_set,
distribution="gaussian",
n.trees=1000,
shrinkage = lambda)
preds <- predict(model, newdata=Biden_test_set, n.trees = 1000)
mse <- mean((Biden_test_set$biden-preds)^2)
mse
}

boost_testing_list <- map_dbl(lambda, boosted_tree_testing)

boost_testing_mse <- as.data.frame(cbind(lambda, boost_testing_list)) %>%
rename(mse = `boost_testing_list`)
boost_testing_mse
```

```
##    lambda      mse
## 1  0.0001 537.1946
## 2  0.0011 405.5798
## 3  0.0021 374.7882
## 4  0.0031 363.5250
## 5  0.0041 356.9523
## 6  0.0051 352.9143
## 7  0.0061 349.9346
## 8  0.0071 347.4808
## 9  0.0081 345.7162
## 10 0.0091 344.6250
## 11 0.0101 343.2759
## 12 0.0111 341.9016
## 13 0.0121 341.3809
## 14 0.0131 339.9537
## 15 0.0141 339.2263
## 16 0.0151 338.4788
## 17 0.0161 337.3454
## 18 0.0171 337.3189
## 19 0.0181 336.6563
## 20 0.0191 335.7410
## 21 0.0201 335.3270
## 22 0.0211 334.6517
## 23 0.0221 333.7779
## 24 0.0231 333.5693
## 25 0.0241 332.6741
## 26 0.0251 332.5580
## 27 0.0261 332.0680
## 28 0.0271 331.7555
## 29 0.0281 330.5119
## 30 0.0291 330.5561
## 31 0.0301 331.8211
## 32 0.0311 330.3657
## 33 0.0321 330.3918
## 34 0.0331 329.4502
## 35 0.0341 329.1992
## 36 0.0351 329.4035
## 37 0.0361 328.7006
## 38 0.0371 328.2107
## 39 0.0381 327.5665
## 40 0.0391 327.2755
```

```
##Plot
ggplot() +
geom_point(data = boost_training_mse, mapping = aes(lambda, mse), color = "hotpink") +
geom_line(data = boost_training_mse, mapping = aes(lambda, mse), color = "hotpink") +
geom_point(data = boost_testing_mse, mapping = aes(lambda, mse), color = "darkcyan") +
geom_line(data = boost_testing_mse, mapping = aes(lambda, mse), color = "darkcyan") +
labs(title = "Mean Squared Errors Across Shrinkage Values",
y = "MSE",
x = "Lambda",
subtitle = "Training (Pink), Testing (Dark Cyan)")
```



4. (10 points) The test MSE values are insensitive to some precise value of $\lambda$ as long as its small enough. Update the boosting procedure by setting $\lambda$ equal to 0.01 (but still over 1000 trees). Report the test MSE and discuss the results. How do they compare?

```
lambda2<- 0.01

##Training
boosted_tree_training2 <- function(lambda2){
model <- gbm(biden ~ .,
data = Biden_training_set,
distribution="gaussian",
n.trees=1000,
shrinkage = lambda2)
preds <- predict(model, newdata=Biden_training_set, n.trees = 1000)
mse <- mean((Biden_training_set$biden-preds)^2)
mse
}

boost_training_list2 <- map_dbl(lambda2, boosted_tree_training2)

boost_training_mse2 <- as.data.frame(cbind(lambda2, boost_training_list2)) %>%
rename(mse = `boost_training_list2`)
boost_training_mse2
```

```
##   lambda2      mse
## 1    0.01 395.2579
```

```r
##Testing
boosted_tree_testing2 <- function(lambda2){
model <- gbm(biden ~ .,
data = Biden_test_set,
distribution="gaussian",
n.trees=1000,
shrinkage = lambda2)
preds <- predict(model, newdata=Biden_test_set, n.trees = 1000)
mse <- mean((Biden_test_set$biden-preds)^2)
mse
}

boost_testing_list2 <- map_dbl(lambda2, boosted_tree_testing2)

boost_testing_mse2 <- as.data.frame(cbind(lambda2, boost_testing_list2)) %>%
rename(mse = `boost_testing_list2`)
boost_testing_mse2
```

```
##   lambda2      mse
## 1    0.01 343.2547
```

The MSE for the training set is now 380.5575 and the MSE for the testing set is now 389.0379. Therefore, the training set outperforms the testing set.
The new training set MSE fits within the range of MSE values for the original range of lambda values. The new testing set MSE also falls within the range of the MSE values for the original range of lambda values.
5. (10 points) Now apply bagging to the training set. What is the test set MSE for this approach?

```r
bagging_training_set<- randomForest(biden~.,data=Biden_training_set)
predict_bag_train<- predict(bagging_training_set,newdata=Biden_test_set)
mse2<- mean((predict_bag_train-Biden_test_set$biden)^2)
mse2
```

```
## [1] 399.2582
```

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

```r
random_forest_train <- randomForest(biden ~ ., data = Biden_training_set)
set.seed(888)
pred_randomforest_train <- predict(random_forest_train, newdata = Biden_test_set)
mse3<- mean((pred_randomforest_train - Biden_test_set$biden)^2)
mse3
```

```
## [1] 399.5679
```

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

```r
linear_model_train<- lm(biden~., data=Biden_training_set)
set.seed(888)
pred_lm<- predict(linear_model_train, newdata=Biden_test_set)
mse4<- mean((pred_lm- Biden_test_set$biden)^2)
mse4
```

```
## [1] 371.6687
```

8. (5 points) Compare test errors across all fits. Discuss which approach generally fits best and how you concluded this.
   Out of the three additional tests performed, the random forest approach generally fits the data best since it has the lowest MSE value. However, overall it is best to use the boosted approach, which

yielded a MSE value much lower than any of the additional approaches used later in the problem set.

### Support Vector Machines

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

```
dim(OJ)
```

```
## [1] 1070   18
```

```
set.seed(888)
OJ.data=data.frame(OJ)
train<-sample(nrow(OJ.data),800)
training_set<- OJ.data[train,]
test_set<- OJ.data[-train,]
```

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.

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

```
##
## Call:
## svm(formula = Purchase ~ ., data = training_set, kernel = "linear",
##     cost = 0.01, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  617
##
##  ( 309 308 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

There were 617 support vectors. 309 in one class and 307 in the other. So we are indeed predicting 2 classes. 3. (5 points) Display the confusion matrix for the classification solution, and also report both the training and test set error rates.

```
pred_test <- predict(svmfit, newdata = test_set[,-1])
confusion_matrix <- table(test_set[, 1], pred_test)
confusion_matrix
```

```
##     pred_test
##       CH  MM
##   CH 150  13
##   MM  57  50
```

```r
#Testing Set Error Rate
classification_error_test <- predict(svmfit,newdata=test_set)
class.table(obs=test_set$Purchase,pred=classification_error_test)
```

```
##     pred
## obs    CH   MM
##   CH 92.0  8.0
##   MM 53.3 46.7
## overall: 74.1
```

```r
#Training Set Error Rate
classification_error_train <- predict(svmfit,newdata=training_set)
class.table(obs=training_set$Purchase,pred=classification_error_train)
```

```
##     pred
## obs    CH   MM
##   CH 91.8  8.2
##   MM 43.9 56.1
## overall: 78
```

In the testing set, we're getting 74.1% correctly classified. In the training set we're correctly classifying 78%. 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
svm_tune<-tune(svm, Purchase~., data= training_set,kernel="linear",
               ranges=list(cost=c(500, 510, 520, 530, 540, 550)))

summary(svm_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   510
##
## - best performance: 0.18
##
## - Detailed performance results:
##   cost   error dispersion
## 1  500 0.18125 0.07574858
## 2  510 0.18000 0.07710022
## 3  520 0.18000 0.07710022
## 4  530 0.18125 0.07574858
## 5  540 0.18000 0.07710022
## 6  550 0.18000 0.07710022
```

The optimal cost is at 500. 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.)

```
best.train.pred <- predict(svm_tune$best.model,newdata=training_set)
class.table(obs=training_set$Purchase,pred=best.train.pred)
```

```
##     pred
## obs    CH   MM
##   CH 88.4 11.6
##   MM 23.9 76.1
## overall: 83.6
```

```
best.test.pred<- predict(svm_tune$best.model,newdata=test_set)
class.table(obs=test_set$Purchase, pred=best.test.pred)
```

```
##     pred
## obs  CH MM
##   CH 92  8
##   MM 29 71
## overall: 83.7
```

The trianing set now correctly classifies 83.6% of the time and the testing set now correctly classifies 83.7% of the time. Both are an improvement over the previous classification rates. My optimal classifier therefore outperformed the original classifier of 0.01.