Problem Set 3

Seth Harrison

2/17/2020

```
library(tidyverse)
library(gbm)
library(rsample)
library(randomForest)
library(e1071)
library(ISLR)
library(caret)
```

Decision Trees

Set Up

```
set.seed(1)

nes2008 <- read_csv("./data/nes2008.csv")

noBiden <- nes2008 %>%
            select(-nes2008$biden)

p <- ncol(noBiden)

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

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.

```
set.seed(1)

train_ind <- sample(nrow(nes2008), size = nrow(nes2008)*.75)

train <- nes2008[train_ind,]
test <- nes2008[-train_ind,]
```

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.

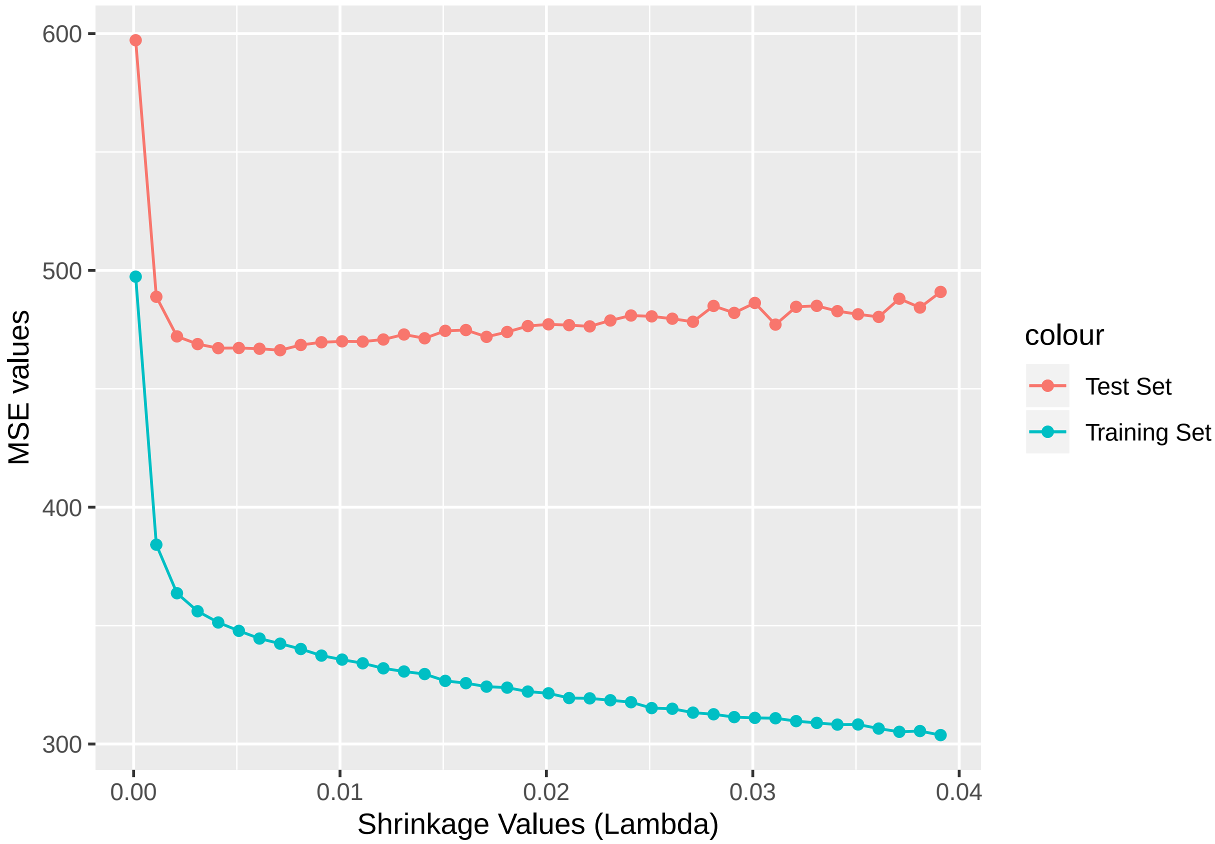
```r
TestMSE <- vector(mode = "numeric", length = length(lambda))
TrainingMSE <- vector(mode = "numeric", length = length(lambda))
for(i in seq_along(lambda)) {
# boosting training set
  boost.train <- gbm(biden ~.,
                   data = train,
                   distribution = "gaussian",
                   n.trees = 1000,
                   shrinkage = lambda[i],
                   interaction.depth = 4
                   )
  training.pred <- predict(boost.train, newdata = train, n.trees = 1000)
  training.mse <- Metrics::mse(training.pred, train$biden)
# predict on test set
  test.pred <- predict(boost.train, newdata = test, n.trees = 1000)
  test.mse <- Metrics::mse(test.pred, test$biden)
# extract MSE and lambda
  TrainingMSE[i] <- training.mse
  TestMSE[i] <- test.mse
  result <- cbind(lambda, TrainingMSE, TestMSE)
  result <- result %>%
    as_tibble()
}
#Plot
result %>%
  ggplot(aes(x = lambda)) +
  geom_point(aes(y = TrainingMSE, color = "Training Set")) +
```

```r
  geom_point(aes(y = TestMSE, color = "Test Set")) +

  geom_line(aes(y = TrainingMSE, color = "Training Set")) +

  geom_line(aes(y = TestMSE, color = "Test Set")) +

labs(x = "Shrinkage Values (Lambda)", y = "MSE values")
```



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
boost.train2 <- gbm(biden ~.,

                   data = train,

                   distribution = "gaussian",

                   n.trees = 1000,

                   shrinkage = lambda[1]*100,

                   interaction.depth = 4
```

```r
                   )


training.pred2 <- predict(boost.train2, newdata = train, n.trees = 1000)

training.mse2 <- Metrics::mse(training.pred2, train$biden)

# predict on test set

test.pred2 <- predict(boost.train2, newdata = test, n.trees = 1000)

test.mse2 <- Metrics::mse(test.pred2, test$biden)

# extract MSE and lambda

TrainingMSE2 <- training.mse2

TestMSE2 <- test.mse2

result <- cbind(lambda, TrainingMSE2, TestMSE2)

result <- result %>%
            as_tibble()

TestMSE2
```

```
## [1] 470.9239
```

The MSE changes only marginally once lambda became greater than .002.

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

```r
bag_biden <- randomForest(data = train,
                          x = train[,2:6],
                          y = train$biden,
                          mtry = p)

# predict on test set

test.predbag <- predict(bag_biden, newdata = test)

test.msebag <- Metrics::mse(test.predbag, test$biden)

test.msebag
```

```
## [1] 550.5081
```

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

```r
rf_biden <- randomForest(data = train,
                         x = train [,2:6],
```

```
                        y = train$biden)

# predict on test set

test.predrf <- predict(rf_biden, newdata = test)

test.mserf <- Metrics::mse(test.predrf, test$biden)

test.mserf
```

```
## [1] 475.1519
```

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

```
lm_biden <- glm(biden~female+age+educ+dem+rep, data = train)

# predict on test set

test.predlm <- predict(lm_biden, newdata = test)

test.mselm <- Metrics::mse(test.predlm, test$biden)

test.mselm
```

```
## [1] 469.9226
```

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

```
mse.table <- cbind(test.mse,test.mse2,test.msebag, test.mserf, test.mselm)

mse.table
```

```
##      test.mse test.mse2 test.msebag test.mserf test.mselm
## [1,] 490.9084  470.9239    550.5081   475.1519   469.9226
```

In terms of test MSE, the boosted MSE with lambda at 0.01 did only marginally better than a simple linear regression. This is because boosting and its subsequent derivations provide a sequence of coefficient vectors, which is the functional form of a linear regression, which minimizes the sum of square errors.

Support Vector Machines

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

```
set.seed(1)

OJ <- ISLR::OJ

OJtrain_ind <- sample(nrow(OJ), size = 800)
```

```r
OJtrain <- OJ[OJtrain_ind,]
OJtest <- OJ[-OJtrain_ind,]
```

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

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJtrain, kernel = "linear",
##     cost = 0.01, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  615
##
##  ( 306 309 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

There were 615 support vectors; 306 in one class and 309 in the other.

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

```r
train.prediction <- predict(svmfit, OJtrain)

confusionMatrix(train.prediction, OJtrain$Purchase, dnn = c("Prediction", "Reference"))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 473 189
##         MM  21 117
##
##                Accuracy : 0.7375
##                  95% CI : (0.7055, 0.7677)
```

```
##     No Information Rate : 0.6175
##     P-Value [Acc > NIR] : 5.044e-13
##
##                   Kappa : 0.3795
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9575
##             Specificity : 0.3824
##          Pos Pred Value : 0.7145
##          Neg Pred Value : 0.8478
##              Prevalence : 0.6175
##          Detection Rate : 0.5913
##    Detection Prevalence : 0.8275
##       Balanced Accuracy : 0.6699
##
##        'Positive' Class : CH
##
```

```r
test.prediction <- predict(svmfit, OJtest)

confusionMatrix(test.prediction, OJtest$Purchase, dnn = c("Prediction", "Reference") )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 156  68
##         MM   3  43
##
##                Accuracy : 0.737
##                  95% CI : (0.6802, 0.7885)
##     No Information Rate : 0.5889
##     P-Value [Acc > NIR] : 2.668e-07
##
##                   Kappa : 0.4043
##
##  Mcnemar's Test P-Value : 3.068e-14
##
##             Sensitivity : 0.9811
##             Specificity : 0.3874
##          Pos Pred Value : 0.6964
##          Neg Pred Value : 0.9348
##              Prevalence : 0.5889
##          Detection Rate : 0.5778
##    Detection Prevalence : 0.8296
##       Balanced Accuracy : 0.6843
##
##        'Positive' Class : CH
##
```

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).

```
tune_c <- tune(svm,
               Purchase  ~ .,
               data = OJtrain,
               kernel = "linear",
               ranges = list(cost = c(0.01, 0.1, 1, 5, 10, 100, 1000)))

summary(tune_c)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.1625
##
## - Detailed performance results:
##    cost   error dispersion
## 1 1e-02 0.16625 0.05138701
## 2 1e-01 0.16250 0.04894725
## 3 1e+00 0.16875 0.04723243
## 4 5e+00 0.16750 0.05041494
## 5 1e+01 0.16500 0.04993051
## 6 1e+02 0.17000 0.05277047
## 7 1e+03 0.17000 0.05277047
```

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.

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

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = OJtrain,
##     ranges = list(cost = c(0.01, 0.1, 1, 5, 10, 100, 1000)),
##     kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##
## Number of Support Vectors:  343
```

```
##
##  ( 171 172 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
train.prediction2 <- predict(tuned_model, OJtrain)

confusionMatrix(train.prediction2, OJtrain$Purchase, dnn = c("Prediction", "Reference"))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 438  71
##         MM  56 235
##
##                Accuracy : 0.8412
##                  95% CI : (0.8141, 0.8659)
##     No Information Rate : 0.6175
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6608
##
##  Mcnemar's Test P-Value : 0.2141
##
##             Sensitivity : 0.8866
##             Specificity : 0.7680
##          Pos Pred Value : 0.8605
##          Neg Pred Value : 0.8076
##              Prevalence : 0.6175
##          Detection Rate : 0.5475
##    Detection Prevalence : 0.6362
##       Balanced Accuracy : 0.8273
##
##        'Positive' Class : CH
##
```

```r
test.prediction2 <- predict(tuned_model, OJtest)

confusionMatrix(test.prediction2, OJtest$Purchase, dnn = c("Prediction", "Reference") )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 140  32
##         MM  19  79
##
##                Accuracy : 0.8111
##                  95% CI : (0.7592, 0.856)
##     No Information Rate : 0.5889
```

```
##     P-Value [Acc > NIR] : 5.479e-15
##
##                   Kappa : 0.6029
##
##  Mcnemar's Test P-Value : 0.09289
##
##             Sensitivity : 0.8805
##             Specificity : 0.7117
##          Pos Pred Value : 0.8140
##          Neg Pred Value : 0.8061
##              Prevalence : 0.5889
##          Detection Rate : 0.5185
##    Detection Prevalence : 0.6370
##       Balanced Accuracy : 0.7961
##
##        'Positive' Class : CH
##
```

The accuracy rates improved by using the optimal cost. Using the standard .01 cost, the accuracy rate on the test set was about 74%, while that improved to about 81% by using the optimally tuned classifier.