Problem Set 3

Pete Cuppernull

2/21/2020

```
knitr::opts_chunk$set(echo = TRUE)
library(tidyverse)
library(here)
library(tree)
library(ISLR)
library(randomForest)
library(MASS)
library(gbm)
library(rpart)
library(ipred)
```

Decision Trees

1. Setup

```
set.seed(1414)
nes <- read_csv(here("data/nes2008.csv"))

p <- nes %>%
  dplyr::select(-biden) %>%
  ncol()

lambda <- seq(0.0001, .04, .001)
```

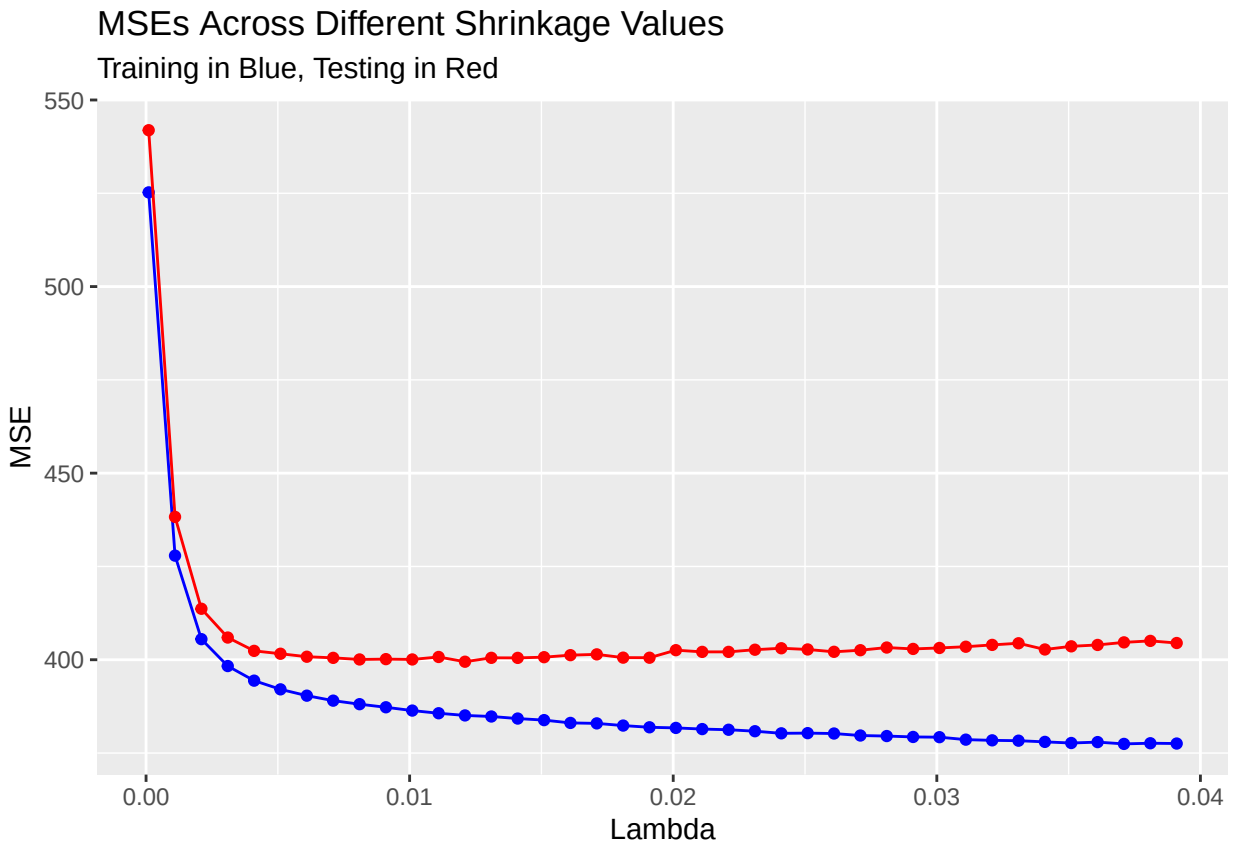
2. Create Training and Test Sets

```
train2 <- sample(1:nrow(nes), (1807*.75))
##I will use all the rows that are NOT in train2
###as my test set later on by subsetting out the training rows
```

3. Boosting w/ 1000 trees

```
##Run Boost on Training data
boosted_tree_train <- function(lambda){
  model <- gbm(biden ~ .,
    data = nes[train2,],
    distribution="gaussian",
```

```
    n.trees=1000,
    shrinkage = lambda)

  preds <- predict(model, newdata=nes[train2,], n.trees = 1000)

  mse <- mean((nes[train2,]$biden-preds)^2)

  mse
}

boost_train_list <- map_dbl(lambda, boosted_tree_train)

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

##Now repeat for test
boosted_tree_test <- function(lambda){
  model <- gbm(biden ~ .,
    data = nes[train2,],
    distribution="gaussian",
    n.trees=1000,
    shrinkage = lambda)

  preds <- predict(model, newdata=nes[-train2,], n.trees = 1000)

  mse <- mean((nes[-train2,]$biden-preds)^2)

  mse
}

boost_test_list <- map_dbl(lambda, boosted_tree_test)

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

ggplot() +
  geom_point(data = boost_train_mses, mapping = aes(lambda, mse), color = "blue") +
  geom_line(data = boost_train_mses, mapping = aes(lambda, mse), color = "blue") +
  geom_point(data = boost_test_mses, mapping = aes(lambda, mse), color = "red") +
  geom_line(data = boost_test_mses, mapping = aes(lambda, mse), color = "red") +
  labs(title = "MSEs Across Different Shrinkage Values",
       y = "MSE",
       x = "Lambda",
       subtitle = "Training in Blue, Testing in Red")
```



Question 4. Fix Lambda

```
mse_01 <- boosted_tree_test(.01)
```

The test MSE for the boosting procedure with a λ of .01 is 400.2883042. This result is approxiamtely in line with our expected resukts, considering the MSEs for λ values of .0091 and .0101 produced in Question 3. The MSE for a λ of .01 is slightly higher than these other values, however, highlighting the importance of estimating models with a range of shrinkage values to identify the ideal model.

Question 5. Apply Bagging

```
biden_bag <- bagging(biden ~ .,
                     data = nes,
                     subset = train2)


bag_preds <- predict(biden_bag, newdata=nes[-train2,])

bag_mse <- mean((nes[-train2,]$biden-bag_preds)^2)
```

The test set MSE for the bagging approach is 402.9984459.

Question 6. Apply Random Forest

```
biden_rf <- randomForest(biden ~ .,
                    data = nes,
                    subset = train2)


rf_preds <- predict(biden_rf, newdata=nes[-train2,])

rf_mse <- mean((nes[-train2,]$biden-rf_preds)^2)
```

The test set MSE for the random forest approach is 409.0675121.

Question 7. Apply Linear Regression

```
biden_lm <- glm(biden ~ .,
                    data = nes,
                    subset = train2)


lm_preds <- predict(biden_lm, newdata=nes[-train2,])

lm_mse <- mean((nes[-train2,]$biden-lm_preds)^2)

lm_mse
```

```
## [1] 399.2977
```

The test set MSE for linear regression is 399.297653.

Question 8

The test errors of the different model fits are as follows:

- Boosted: 399.468513
- Boosted with .01 λ: 400.1461969
- Bagging: 402.9984459
- Random Forest: 409.0675121
- Linear Model: 399.297653

The model which generally fits the best is the linear model, also has a MSE of 399.297653, which is slightly smaller than the boosted model's MSE of 399.468513. It would likely be preferable to use the linear model because of the strong performance and relatively easy interpretation.

Support Vector Machines

Question 1. Create Training and Test Sets

```
train_num <- sample(1:nrow(OJ), 800)
oj_train <- OJ[train_num,]
oj_test <- OJ[-train_num,]
```

Question 2. Fit SV Classifier

```
library(e1071)
svmfit_oj <- svm(Purchase ~ .,
             data = oj_train,
             kernel = "linear",
             cost = .01)
```

The number of support vectors produced by a cost of .01 is 431. This seems to be a realtively large number of support vectors, considering that there are only 800 observations in the training set.

Question 3. Display Confusion Matrix

```
class_pred <- predict(svmfit_oj, oj_test)
confusion_matrix <- table(predicted = class_pred, true = oj_test$Purchase)

test_error <- sum(confusion_matrix[row(confusion_matrix) !=
                                   col(confusion_matrix)]) / sum(confusion_matrix)

class_pred_train <- predict(svmfit_oj, oj_train)
confusion_matrix_train <- table(predicted = class_pred_train, true = oj_train$Purchase)
train_error <- sum(confusion_matrix_train[row(confusion_matrix_train) !=
                            col(confusion_matrix_train)]) / sum(confusion_matrix_train)

confusion_matrix
```

```
##           true
## predicted  CH  MM
##        CH 149  20
##        MM  26  75
```

The training error rate is 0.17 and the test error rate is 0.1703704.

Question 4. Find Optimal Cost

```
tune_oj <- tune(svm,
                Purchase ~ .,
                data = oj_train,
                kernel = "linear",
                ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100, 1000)))

# best?
```

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

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = oj_train,
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100, 1000)),
##     kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  5
##
## Number of Support Vectors:  325
##
##  ( 162 163 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
##Let's re-tune around the 5 value, which did the best
tune_oj2 <- tune(svm,
                 Purchase ~ .,
                 data = oj_train,
                 kernel = "linear",
                 ranges = list(cost = seq(2, 10, .5)))

tuned_model2 <- tune_oj2$best.model
```

The optimal cost of the model is 5.

Question 5. Compute New Training and Test Error Rates for New Cost

```
svmfit_oj_tuned_cost <- svm(Purchase ~ .,
             data = oj_train,
             kernel = "linear",
             cost = .04)


class_pred_tuned <- predict(svmfit_oj_tuned_cost, oj_test)
confusion_matrix_tuned <- table(predicted = class_pred_tuned, true = oj_test$Purchase)

test_error_tuned <- sum(confusion_matrix_tuned[row(confusion_matrix_tuned) !=
                          col(confusion_matrix_tuned)]) / sum(confusion_matrix_tuned)

class_pred_train_tuned <- predict(svmfit_oj_tuned_cost, oj_train)

confusion_matrix_train_tuned <- table(predicted = class_pred_train_tuned,
```

```
                                      true = oj_train$Purchase)

train_error_tuned <- sum(confusion_matrix_train_tuned[row(confusion_matrix_train_tuned) !=
                       col(confusion_matrix_train_tuned)]) / sum(confusion_matrix_train_tuned)

confusion_matrix_tuned
```

```
##           true
## predicted  CH  MM
##         CH 149  19
##         MM  26  76
```

The training set error rate for the tuned model is 0.16125 and the test set error rate for the tuned model is 0.1666667. This is slightly better than the naive model with a cost of .01, which had training and test error rates of 0.17 and 0.1703704, respectively. In substantive terms, this difference is quite minimal – the tuned model correctly classified exactly one observation more than the naive model for the test set (out of 270 observations). It did, however, classify eight more observations correctly than the naive model for the training set. While the performance of the tuned classifier is, at best, marginally better than the naive classifier, I would argue that it is still better to leverage the tuned model because of the minimal resources and time required to achieve a slightly better predictor.