Problem Set 3

Grace Wright

2/10/2020

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

```r
# setting the seed
set.seed(123)

# importing data
nes08 <- read.csv("nes2008.csv")

# total number of features except for the biden feelings
p <- ncol(nes08[-1])

# set lambda (shrinkage/learning rate) from .0001 to .04, by .001
lambda <- seq(from = .0001, to = .04, by = .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.

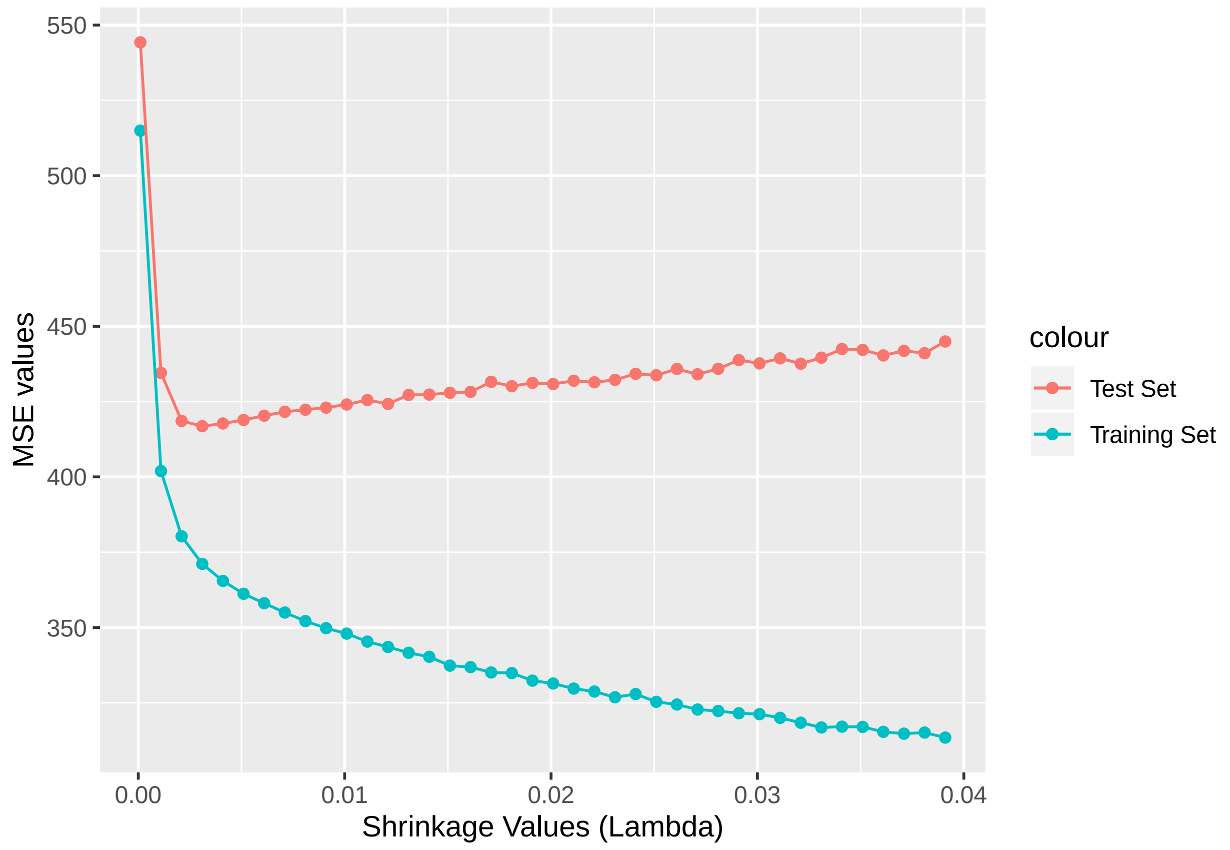
```r
# splitting data into training and test sets
# ??? will this automatically put the remaining observations in test
split <- initial_split(nes08, prop = .75)
train <- training(split)
test  <- testing(split)
```

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
test_mse_storage <- vector(mode = "numeric", length = length(lambda))

train_mse_storage <- 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)


  train.pred <- predict(boost.train, newdata = train, n.trees = 1000)
  train.mse <- Metrics::mse(train.pred, train$biden)

  # making prediction on the test set

  test.pred <- predict(boost.train, newdata = test, n.trees = 1000)
  test.mse <- Metrics::mse(test.pred, test$biden)
  test.mse

  # extract MSE and lambda values

  train_mse_storage[i] <- train.mse
  test_mse_storage[i] <- test.mse
  result <- cbind(lambda, train_mse_storage, test_mse_storage)
  result <- result %>%
    as.tibble()
  print(result)
}
```

```
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001              515.             544.
##  2 0.0011                0                0
##  3 0.0021                0                0
##  4 0.0031                0                0
##  5 0.0041                0                0
##  6 0.0051                0                0
##  7 0.0061                0                0
##  8 0.0071                0                0
##  9 0.0081                0                0
## 10 0.0091                0                0
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001              515.             544.
##  2 0.0011              402.             435.
##  3 0.0021                0                0
##  4 0.0031                0                0
##  5 0.0041                0                0
##  6 0.0051                0                0
##  7 0.0061                0                0
##  8 0.0071                0                0
##  9 0.0081                0                0
## 10 0.0091                0                0
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001              515.             544.
##  2 0.0011              402.             435.
##  3 0.0021              380.             419.
##  4 0.0031                0                0
##  5 0.0041                0                0
##  6 0.0051                0                0
##  7 0.0061                0                0
##  8 0.0071                0                0
##  9 0.0081                0                0
## 10 0.0091                0                0
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001              515.             544.
##  2 0.0011              402.             435.
##  3 0.0021              380.             419.
##  4 0.0031              371.             417.
##  5 0.0041                0               0
##  6 0.0051                0               0
##  7 0.0061                0               0
##  8 0.0071                0               0
##  9 0.0081                0               0
## 10 0.0091                0               0
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001              515.             544.
##  2 0.0011              402.             435.
##  3 0.0021              380.             419.
##  4 0.0031              371.             417.
##  5 0.0041              365.             418.
##  6 0.0051                0               0
##  7 0.0061                0               0
##  8 0.0071                0               0
##  9 0.0081                0               0
## 10 0.0091                0               0
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001              515.             544.
##  2 0.0011              402.             435.
##  3 0.0021              380.             419.
##  4 0.0031              371.             417.
##  5 0.0041              365.             418.
##  6 0.0051              361.             419.
##  7 0.0061                0               0
##  8 0.0071                0               0
##  9 0.0081                0               0
## 10 0.0091                0               0
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001              515.             544.
##  2 0.0011              402.             435.
##  3 0.0021              380.             419.
##  4 0.0031              371.             417.
##  5 0.0041              365.             418.
##  6 0.0051              361.             419.
##  7 0.0061              358.             420.
##  8 0.0071                0               0
##  9 0.0081                0               0
## 10 0.0091                0               0
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001              515.             544.
##  2 0.0011              402.             435.
##  3 0.0021              380.             419.
##  4 0.0031              371.             417.
##  5 0.0041              365.             418.
##  6 0.0051              361.             419.
##  7 0.0061              358.             420.
##  8 0.0071              355.             422.
##  9 0.0081                0                0
## 10 0.0091                0                0
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001              515.             544.
##  2 0.0011              402.             435.
##  3 0.0021              380.             419.
##  4 0.0031              371.             417.
##  5 0.0041              365.             418.
##  6 0.0051              361.             419.
##  7 0.0061              358.             420.
##  8 0.0071              355.             422.
##  9 0.0081              352.             422.
## 10 0.0091                0                0
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001              515.             544.
##  2 0.0011              402.             435.
##  3 0.0021              380.             419.
##  4 0.0031              371.             417.
##  5 0.0041              365.             418.
##  6 0.0051              361.             419.
##  7 0.0061              358.             420.
##  8 0.0071              355.             422.
##  9 0.0081              352.             422.
## 10 0.0091              350.             423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##     lambda train_mse_storage test_mse_storage
##      <dbl>             <dbl>            <dbl>
##  1 0.0001              515.             544.
##  2 0.0011              402.             435.
##  3 0.0021              380.             419.
##  4 0.0031              371.             417.
##  5 0.0041              365.             418.
##  6 0.0051              361.             419.
##  7 0.0061              358.             420.
##  8 0.0071              355.             422.
##  9 0.0081              352.             422.
## 10 0.0091              350.             423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##     lambda train_mse_storage test_mse_storage
##      <dbl>             <dbl>            <dbl>
##  1 0.0001              515.             544.
##  2 0.0011              402.             435.
##  3 0.0021              380.             419.
##  4 0.0031              371.             417.
##  5 0.0041              365.             418.
##  6 0.0051              361.             419.
##  7 0.0061              358.             420.
##  8 0.0071              355.             422.
##  9 0.0081              352.             422.
## 10 0.0091              350.             423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##     lambda train_mse_storage test_mse_storage
##      <dbl>             <dbl>            <dbl>
##  1 0.0001              515.             544.
##  2 0.0011              402.             435.
##  3 0.0021              380.             419.
##  4 0.0031              371.             417.
##  5 0.0041              365.             418.
##  6 0.0051              361.             419.
##  7 0.0061              358.             420.
##  8 0.0071              355.             422.
##  9 0.0081              352.             422.
## 10 0.0091              350.             423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##     lambda train_mse_storage test_mse_storage
##     <dbl>            <dbl>           <dbl>
##  1 0.0001             515.            544.
##  2 0.0011             402.            435.
##  3 0.0021             380.            419.
##  4 0.0031             371.            417.
##  5 0.0041             365.            418.
##  6 0.0051             361.            419.
##  7 0.0061             358.            420.
##  8 0.0071             355.            422.
##  9 0.0081             352.            422.
## 10 0.0091             350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>            <dbl>           <dbl>
##  1 0.0001             515.            544.
##  2 0.0011             402.            435.
##  3 0.0021             380.            419.
##  4 0.0031             371.            417.
##  5 0.0041             365.            418.
##  6 0.0051             361.            419.
##  7 0.0061             358.            420.
##  8 0.0071             355.            422.
##  9 0.0081             352.            422.
## 10 0.0091             350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>            <dbl>           <dbl>
##  1 0.0001             515.            544.
##  2 0.0011             402.            435.
##  3 0.0021             380.            419.
##  4 0.0031             371.            417.
##  5 0.0041             365.            418.
##  6 0.0051             361.            419.
##  7 0.0061             358.            420.
##  8 0.0071             355.            422.
##  9 0.0081             352.            422.
## 10 0.0091             350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>            <dbl>           <dbl>
##  1 0.0001             515.            544.
##  2 0.0011             402.            435.
##  3 0.0021              380.            419.
##  4 0.0031              371.            417.
##  5 0.0041              365.            418.
##  6 0.0051              361.            419.
##  7 0.0061              358.            420.
##  8 0.0071              355.            422.
##  9 0.0081              352.            422.
## 10 0.0091              350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001              515.            544.
##  2 0.0011              402.            435.
##  3 0.0021              380.            419.
##  4 0.0031              371.            417.
##  5 0.0041              365.            418.
##  6 0.0051              361.            419.
##  7 0.0061              358.            420.
##  8 0.0071              355.            422.
##  9 0.0081              352.            422.
## 10 0.0091              350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001              515.            544.
##  2 0.0011              402.            435.
##  3 0.0021              380.            419.
##  4 0.0031              371.            417.
##  5 0.0041              365.            418.
##  6 0.0051              361.            419.
##  7 0.0061              358.            420.
##  8 0.0071              355.            422.
##  9 0.0081              352.            422.
## 10 0.0091              350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001              515.            544.
##  2 0.0011              402.            435.
##  3 0.0021              380.            419.
##  4 0.0031              371.            417.
##  5 0.0041              365.            418.
##  6 0.0051              361.            419.
##  7 0.0061              358.            420.
##  8 0.0071              355.            422.
##  9 0.0081              352.            422.
## 10 0.0091              350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##     lambda train_mse_storage test_mse_storage
##      <dbl>             <dbl>            <dbl>
##  1 0.0001              515.            544.
##  2 0.0011              402.            435.
##  3 0.0021              380.            419.
##  4 0.0031              371.            417.
##  5 0.0041              365.            418.
##  6 0.0051              361.            419.
##  7 0.0061              358.            420.
##  8 0.0071              355.            422.
##  9 0.0081              352.            422.
## 10 0.0091              350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##     lambda train_mse_storage test_mse_storage
##      <dbl>             <dbl>            <dbl>
##  1 0.0001              515.            544.
##  2 0.0011              402.            435.
##  3 0.0021              380.            419.
##  4 0.0031              371.            417.
##  5 0.0041              365.            418.
##  6 0.0051              361.            419.
##  7 0.0061              358.            420.
##  8 0.0071              355.            422.
##  9 0.0081              352.            422.
## 10 0.0091              350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##     lambda train_mse_storage test_mse_storage
##      <dbl>             <dbl>            <dbl>
##  1 0.0001              515.            544.
##  2 0.0011              402.            435.
##  3 0.0021              380.            419.
##  4 0.0031              371.            417.
##  5 0.0041              365.            418.
##  6 0.0051              361.            419.
##  7 0.0061              358.            420.
##  8 0.0071              355.            422.
##  9 0.0081             352.            422.
## 10 0.0091             350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001             515.            544.
##  2 0.0011             402.            435.
##  3 0.0021             380.            419.
##  4 0.0031             371.            417.
##  5 0.0041             365.            418.
##  6 0.0051             361.            419.
##  7 0.0061             358.            420.
##  8 0.0071             355.            422.
##  9 0.0081             352.            422.
## 10 0.0091             350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001             515.            544.
##  2 0.0011             402.            435.
##  3 0.0021             380.            419.
##  4 0.0031             371.            417.
##  5 0.0041             365.            418.
##  6 0.0051             361.            419.
##  7 0.0061             358.            420.
##  8 0.0071             355.            422.
##  9 0.0081             352.            422.
## 10 0.0091             350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001             515.            544.
##  2 0.0011             402.            435.
##  3 0.0021             380.            419.
##  4 0.0031             371.            417.
##  5 0.0041             365.            418.
##  6 0.0051             361.            419.
##  7 0.0061             358.            420.
##  8 0.0071             355.            422.
##  9 0.0081             352.            422.
## 10 0.0091             350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001              515.             544.
##  2 0.0011              402.             435.
##  3 0.0021              380.             419.
##  4 0.0031              371.             417.
##  5 0.0041              365.             418.
##  6 0.0051              361.             419.
##  7 0.0061              358.             420.
##  8 0.0071              355.             422.
##  9 0.0081              352.             422.
## 10 0.0091              350.             423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001              515.             544.
##  2 0.0011              402.             435.
##  3 0.0021              380.             419.
##  4 0.0031              371.             417.
##  5 0.0041              365.             418.
##  6 0.0051              361.             419.
##  7 0.0061              358.             420.
##  8 0.0071              355.             422.
##  9 0.0081              352.             422.
## 10 0.0091              350.             423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001              515.             544.
##  2 0.0011              402.             435.
##  3 0.0021              380.             419.
##  4 0.0031              371.             417.
##  5 0.0041              365.             418.
##  6 0.0051              361.             419.
##  7 0.0061              358.             420.
##  8 0.0071              355.             422.
##  9 0.0081              352.             422.
## 10 0.0091              350.             423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001            515.            544.
##  2 0.0011            402.            435.
##  3 0.0021            380.            419.
##  4 0.0031            371.            417.
##  5 0.0041            365.            418.
##  6 0.0051            361.            419.
##  7 0.0061            358.            420.
##  8 0.0071            355.            422.
##  9 0.0081            352.            422.
## 10 0.0091            350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001            515.            544.
##  2 0.0011            402.            435.
##  3 0.0021            380.            419.
##  4 0.0031            371.            417.
##  5 0.0041            365.            418.
##  6 0.0051            361.            419.
##  7 0.0061            358.            420.
##  8 0.0071            355.            422.
##  9 0.0081            352.            422.
## 10 0.0091            350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001            515.            544.
##  2 0.0011            402.            435.
##  3 0.0021            380.            419.
##  4 0.0031            371.            417.
##  5 0.0041            365.            418.
##  6 0.0051            361.            419.
##  7 0.0061            358.            420.
##  8 0.0071            355.            422.
##  9 0.0081            352.            422.
## 10 0.0091            350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001            515.            544.
##  2 0.0011            402.            435.
##  3 0.0021            380.            419.
##  4 0.0031            371.            417.
##  5 0.0041            365.            418.
##  6 0.0051            361.            419.
##  7 0.0061            358.            420.
##  8 0.0071            355.            422.
##  9 0.0081            352.            422.
## 10 0.0091            350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001            515.            544.
##  2 0.0011            402.            435.
##  3 0.0021            380.            419.
##  4 0.0031            371.            417.
##  5 0.0041            365.            418.
##  6 0.0051            361.            419.
##  7 0.0061            358.            420.
##  8 0.0071            355.            422.
##  9 0.0081            352.            422.
## 10 0.0091            350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001            515.            544.
##  2 0.0011            402.            435.
##  3 0.0021            380.            419.
##  4 0.0031            371.            417.
##  5 0.0041            365.            418.
##  6 0.0051            361.            419.
##  7 0.0061            358.            420.
##  8 0.0071            355.            422.
##  9 0.0081            352.            422.
## 10 0.0091            350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001            515.            544.
##  2 0.0011            402.            435.
##  3 0.0021            380.            419.
##  4 0.0031            371.            417.
##  5 0.0041            365.            418.
##  6 0.0051            361.            419.
##  7 0.0061             358.            420.
##  8 0.0071             355.            422.
##  9 0.0081             352.            422.
## 10 0.0091             350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001             515.            544.
##  2 0.0011             402.            435.
##  3 0.0021             380.            419.
##  4 0.0031             371.            417.
##  5 0.0041             365.            418.
##  6 0.0051             361.            419.
##  7 0.0061             358.            420.
##  8 0.0071             355.            422.
##  9 0.0081             352.            422.
## 10 0.0091             350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001             515.            544.
##  2 0.0011             402.            435.
##  3 0.0021             380.            419.
##  4 0.0031             371.            417.
##  5 0.0041             365.            418.
##  6 0.0051             361.            419.
##  7 0.0061             358.            420.
##  8 0.0071             355.            422.
##  9 0.0081             352.            422.
## 10 0.0091             350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001             515.            544.
##  2 0.0011             402.            435.
##  3 0.0021             380.            419.
##  4 0.0031             371.            417.
##  5 0.0041             365.            418.
##  6 0.0051             361.            419.
##  7 0.0061             358.            420.
##  8 0.0071             355.            422.
##  9 0.0081             352.            422.
## 10 0.0091              350.            423.
## # ... with 30 more rows
## # A tibble: 40 x 3
##    lambda train_mse_storage test_mse_storage
##     <dbl>             <dbl>            <dbl>
##  1 0.0001              515.            544.
##  2 0.0011              402.            435.
##  3 0.0021              380.            419.
##  4 0.0031              371.            417.
##  5 0.0041              365.            418.
##  6 0.0051              361.            419.
##  7 0.0061              358.            420.
##  8 0.0071              355.            422.
##  9 0.0081              352.            422.
## 10 0.0091              350.            423.
## # ... with 30 more rows
```

```
result %>%
  ggplot(aes(x = lambda)) +
  geom_point(aes(y = train_mse_storage, color = "Training Set")) +
  geom_point(aes(y = test_mse_storage, color = "Test Set")) +
  geom_line(aes(y = train_mse_storage , color = "Training Set")) +
  geom_line(aes(y = test_mse_storage, color = "Test Set")) +
  labs(x = "Shrinkage Values (Lambda)", y = "MSE values")
```



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?

```
for(i in seq_along(lambda)){
# boosting training set
boost.train2 <- gbm(biden ~.,
                    data = train,
                    distribution = "gaussian",
                    n.trees = 1000,
                    shrinkage = .01,
                    interaction.depth = 4)


  train.pred2 <- predict(boost.train, newdata = train, n.trees = 1000)
  train.mse2 <- Metrics::mse(train.pred, train$biden)

  # making prediction on the test set
  test.pred2 <- predict(boost.train, newdata = test, n.trees = 1000)
  test.mse2 <- Metrics::mse(test.pred, test$biden)

  # extract MSE and lambda values
}
 test.mse2
```

```
## [1] 444.9613
```

When the shrinkage (lambda), or learning, rate is set to 0.01 in the boosting approach the mean squared error is 445.9341. When the lambda is set to a continuous variable increasing by a value of 0.001, the values are almost identical to before, with the mean squared error equal to 445.9518. This is because when the learner, or shrinkage, rate is very small (set above as lambda = .01 or .001), the differences in the results are negligible because the learner rate is already significantly small and slow. Additionally, graphically we can see that when the lambda values move closer to zero the mean squared error values sharply increase, and the MSE levels out as lambda values increase. This implies that the model MSE is generally insensitive to larger lambda values.

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

```
bagging <- randomForest(biden ~ .,
                  data = train,
                  mtry = p)
bagging.test.pred <- predict(bagging, newdata = test)
bagging.test.mse <- Metrics::mse(bagging.test.pred, test$biden)
bagging.test.mse
```

```
## [1] 505.3537
```

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

```
rf <- randomForest(biden ~ .,
                  data = train)
rf.test.pred <- predict(rf, newdata = test)
rf.test.mse <- Metrics::mse(rf.test.pred, test$biden)
rf.test.mse
```

```
## [1] 424.2423
```

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

```
lm_train <- lm(biden ~ .,
               data = train)
summary(lm_train)
```

```
##
## Call:
## lm(formula = biden ~ ., data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -76.084 -11.094   0.807  12.927  53.365
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  59.17586    3.58552  16.504  < 2e-16 ***
## female        4.35733    1.08803   4.005 6.54e-05 ***
## age           0.03851    0.03240   1.188   0.2349
## educ         -0.38283    0.22369  -1.711   0.0872 .
## dem          16.06643    1.22649  13.100  < 2e-16 ***
## rep         -14.49164    1.48497  -9.759  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 19.77 on 1350 degrees of freedom
## Multiple R-squared:  0.283,  Adjusted R-squared:  0.2803
## F-statistic: 106.6 on 5 and 1350 DF,  p-value: < 2.2e-16
```

```
lm.test.pred <- predict(lm_train, newdata = test)
lm.test.mse <- Metrics::mse(lm.test.pred, test$biden)
summary(lm.test.mse)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   414.3   414.3   414.3   414.3   414.3   414.3
```

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

Based on the results, it seems like the linear regression model is the best fit becasue it has the lowest mean squared error (mse = 414.3). The random forest approach has the next best fit with a mean squared error (mse) of 422.597. The boosting approach is the next best fit with a mean squared error of 445.9341 when the lambda (or shrinkage rate) is set to 0.01. When the lambda, or shrinkage rate, for the boosting method is set to a continuous variable increasing by a value of 0.001, the values are almost identical to before, with the mean squared error equal to 445.9518. This is because when the learner, or shrinkage, rate is very small as set above (lambda = .01 or .001), the differences in the results are negligible because the learner rate is already significantly small and slow. Additionally, graphically we can see that when the lambda values move closer to zero the mean squared error values sharply increase. Finally, the worst fit was in the bagging method with a mean squared error of 505.9228. This method is similar to the random forests, but has an "mtry" value set to p, and in this case p = 5. The default mtry value for the randomForest function is p/3, thus the mtry value is less in the random forest approach than in the bagging approach above. This shows us that an increased mtry value, in this case, results in an increased mean squared error. The bagging and random forest approaches having the worst fit also makes sense because they are non-parametric meaning they make less assumptions about the data than a parametric approach, like linear regression would make. The less assumptions

Support Vector Machines

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

```r
# setting the seed
set.seed(1234)
# importing data

OJ$Purchase <- factor(OJ$Purchase)

OJ_split <- initial_split(OJ, prop=0.747663)
train <- training(OJ_split)
test <- testing(OJ_split)
```

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
#SVM classifier
svm1 <- svm(Purchase ~ .,
            data = train,
            kernel = "linear",
            cost = 0.01,
            scale = FALSE)
rm(svm)
summary(svm1)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "linear",
##     cost = 0.01, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  613
##
##  ( 308 305 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

The results of the support vector machine tells us that when the data is enlarged into a higher dimensional space it requires many support vectors (613) in order to get the proper separation of data. The support vector classifier fits well for the data. Additionally, it requres two classes and levels.

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

```r
#Generate predicted values for train and test sets
Purchase1 <- predict(svm1, train)
Purchase2 <- predict(svm1, test)

#Confusion matrix for training set predictions
table(predicted = Purchase1,
      true = train$Purchase)
```

```
##          true
## predicted  CH  MM
##        CH 433 126
##        MM  57 184
```

```
#Error rate, calculated as misclassified/total
(131+59)/800
```

```
## [1] 0.2375
```

```
#Confusion matrix for test set predictions
table(predicted = Purchase2,
      true = test$Purchase)
```

```
##          true
## predicted  CH  MM
##        CH 149  55
##        MM  14  52
```

```
#Error rate, calculated as misclassified/total
(35+10)/270
```

```
## [1] 0.1666667
```

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).

```
#Use tuning function to test range of cost values
tune.c <- tune(method = svm,
                train.x = Purchase ~ .,
                data = train,
               kernel = "linear",
                ranges = list(cost = c(0.01, 0.1, 1, 10, 100, 1000)))

#Subset best model and look at summary to identify its cost value
tuned.model <- tune.c$best.model
summary(tuned.model)
```

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

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
#Generate predicted values for train and test sets
Purchase3 <- predict(tuned.model, train)
Purchase4 <- predict(tuned.model, test)

#Confusion matrix for training set predictions
table(predicted = Purchase3,
      true = train$Purchase)
```

```
##          true
## predicted  CH  MM
##        CH 432  71
##        MM  58 239
```

```
#Error rate, calculated as misclassified / total
(83+59)/800
```

```
## [1] 0.1775
```

```
#Confusion matrix for test set predictions
table(predicted = Purchase4,
      true = test$Purchase)
```

```
##          true
## predicted  CH  MM
##        CH 140  28
##        MM  23  79
```

```
#Error rate, calculated as misclassified / total
(23+12)/270
```

```
## [1] 0.1296296
```

The error rates for the optimal training and test sets are fairly comparable (train error rate = 0.1775; test error rate = 0.1296). As noted by the slightly smaller value in the test set, the test set performed slightly better. As expected, if one compares the optimal training and test error rates to the original training and test error rates, the optimal error rates are smaller, and thus better, in both the training and test sets. This makes sense as the purpose of the tuning function for the optimal model was to determine the cost value which results in the most accurate classifier. However, one concern with the optimally tuned classifier is overfitting, or the model having a misleadingly good fit because it was too finely "tuned".