PS3

Soowon Jo

2/15/2020

##Problem Set 3: Trees & Machines

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

##   Attaching packages                                           tidyverse 1.3.0

##   ggplot2 3.2.1      purrr   0.3.3
##   tibble  2.1.3      stringr 1.4.0
##   tidyr   1.0.0      forcats 0.4.0
##   readr   1.3.1

##   Conflicts                                                tidyverse_conflicts()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift

## Loaded gbm 2.1.5

## Registered S3 method overwritten by 'tree':
##   method     from
##   print.tree cli
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:dplyr':
##
##     combine
```

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(210)
setwd("/Users/soowonjo/Desktop/MachineLearning/PB3")
data = read.csv("nes2008.csv")
p <- ncol(data[-1])
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 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.

```
smp_size <- floor(0.75 * nrow(data))

set.seed(110)
train_ind <- sample(seq_len(nrow(data)), size = smp_size)

trainset <- data[train_ind, ]
testset <- data[-train_ind, ]
```

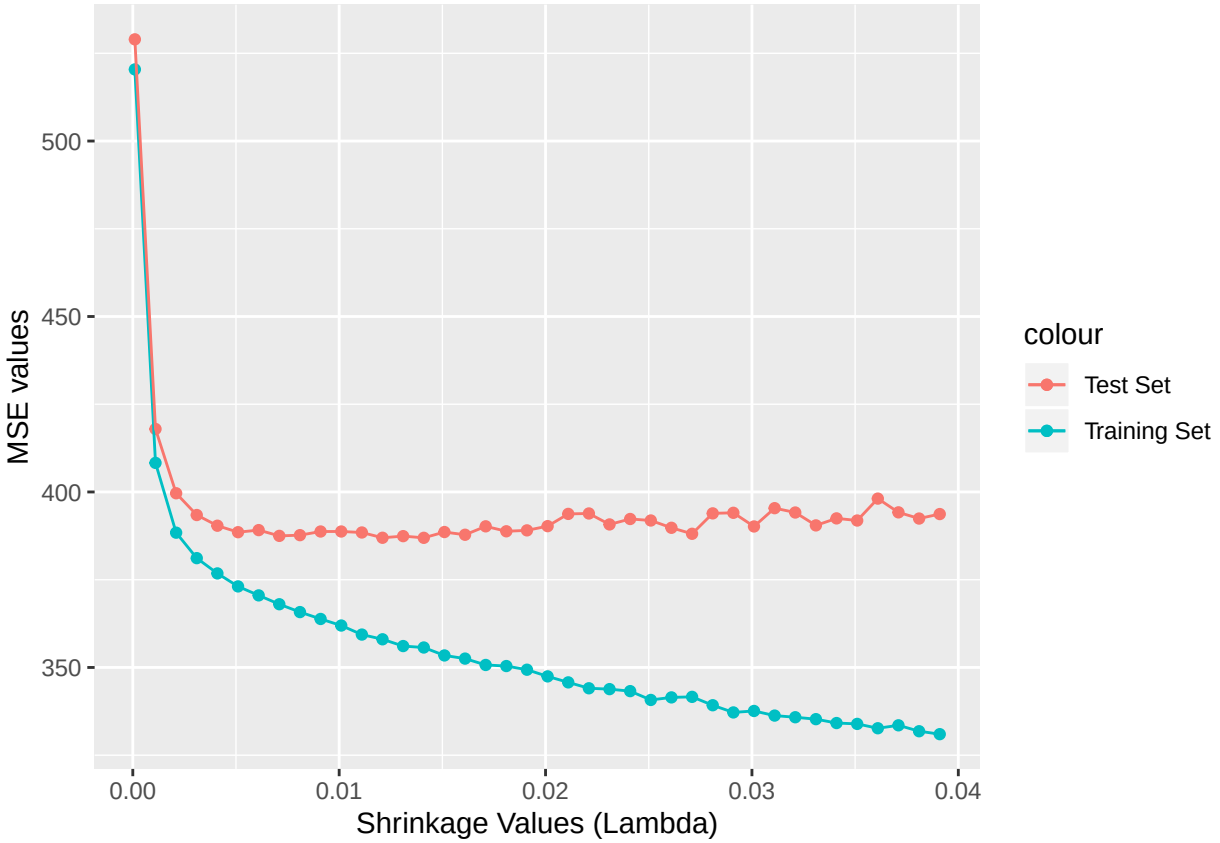
####3. (15 points) Create empty objects to store training and testing MSE, and then write a loop to perform boosting on the training set with 1,000 trees for the pre-defined range of values of the shrinkage parameter, lambda( ). Then, plot the training set and test set MSE across shrinkage values.

```
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 = trainset,
    distribution = "gaussian",
    n.trees = 1000,
    shrinkage = lambda[i],
    interaction.depth = 4)

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

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

# extract MSE and lambda values
trainingMSE[i] <- training.mse
testMSE[i] <- test.mse

result <- cbind(lambda, trainingMSE, testMSE)
result <- result %>%
    as.tibble()
}
```

```
## Warning: `as.tibble()` is deprecated, use `as_tibble()` (but mind the new semantics).
## This warning is displayed once per session.
```

```
result %>%
    ggplot(aes(x = lambda)) +
    geom_point(aes(y = trainingMSE, color = "Training Set")) +
    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")
```



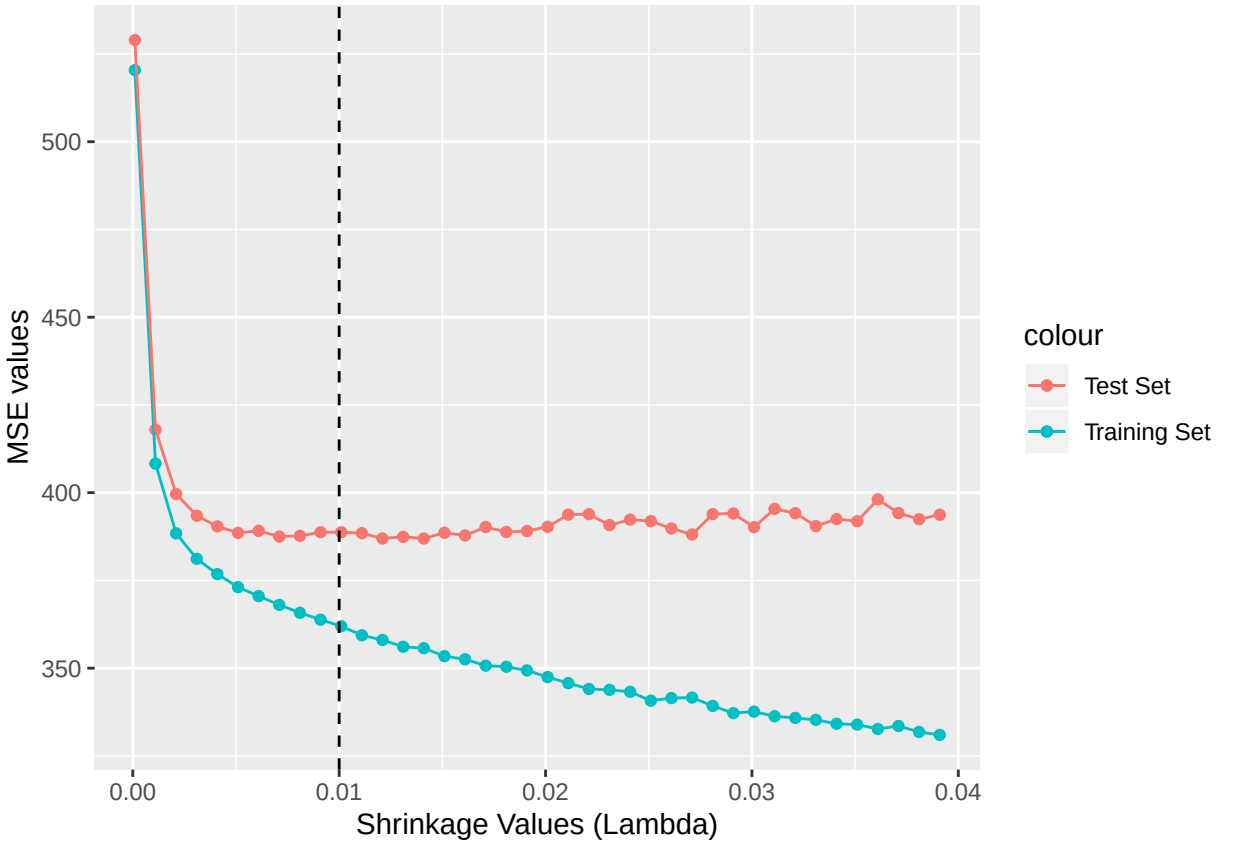
####4. (10 points) The test MSE values are insensitive to some precise value of lambda as long as its small enough. Update the boosting procedure by setting lambda equal to 0.01 (but still over 1000 trees). Report the test MSE and discuss the results. How do they compare?

```
boost.train <- gbm(biden ~.,
    data = trainset,
    distribution = "gaussian",
    n.trees = 1000,
    shrinkage = 0.01,
    interaction.depth = 4)

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

# report the test MSE
test.mse_new
```

```
## [1] 388.9172
```

```
result %>%
    ggplot(aes(x = lambda)) +
    geom_point(aes(y = trainingMSE, color = "Training Set")) +
    geom_point(aes(y = testMSE, color = "Test Set")) +
    geom_line(aes(y = trainingMSE, color = "Training Set")) +
```

```
    geom_line(aes(y = testMSE, color = "Test Set")) +
    geom_vline(xintercept=0.01, linetype="dashed")+
    labs(x = "Shrinkage Values (Lambda)", y = "MSE values")
```



The test MSE for lambda equal to 0.01 is 388.9172. Based on the plot above, the MSE appears roughly the same as lambda increases. When the value of lambda is smaller than 0.01, the models have much larger test MSEs, which means that the model test MSE is insensitive to shrinkage values once they are larger than 0.01.

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

```
bagging <- randomForest(biden ~ .,
    data = trainset,
    mtry = p)

bagging.test.pred <- predict(bagging, newdata = testset)
bagging.test.mse <- Metrics::mse(bagging.test.pred, testset$biden)
bagging.test.mse
```

```
## [1] 451.3808
```

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

```
rf <- randomForest(biden ~ .,
                   data = trainset,
                   importance = TRUE)

rf.test.pred <- predict(rf, newdata = testset)
rf.test.mse <- Metrics::mse(rf.test.pred, testset$biden)
rf.test.mse
```

```
## [1] 402.4014
```

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

```
lm_train <- glm(biden ~ .,data = trainset)
lm.test.pred <- predict(lm_train, newdata = testset)
lm.test.mse <- Metrics::mse(lm.test.pred, testset$biden)
lm.test.mse
```

```
## [1] 392.3925
```

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

The boosting approach results in the lowest MSE across all fits (388.9172) meaning that this approach fits best. The test MSEs received from the boosting and linear regression approaches were similar around 390. The fit with the bagging approach with a lambda parameter of at least 0.01, on the other hand, has the largest test error (451.3808) making it the worst fit of all methods. The bagging approach has an "mtry" values set to p = 5. The mtry value in the bagging approach is larger than that in the randomForest function (p/3). When comparing the error values of both randomforest and bagging approaches, this implies that an increased mtry value results in an increased mean squared error.

Support Vector Machines

####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(nrow(OJ),800)
OJ.train = OJ[train,]
OJ.test = OJ[-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 = 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
```

The result indicates that more support vectors (623) are required to get the proper separation of data when the data is enlarged into a higher dimensional space. Moreover, it requires 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.

```
#Generate predicted values for train and test sets
Purchase1 <- predict(svmfit, OJ.train)
Purchase2 <- predict(svmfit, OJ.test)

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

```
##          true
## predicted  CH  MM
##        CH 466 177
##        MM  22 135
```

```
#Error rate, calculated as misclassified/total
(177+22)/800
```

```
## [1] 0.24875
```

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

```
##          true
## predicted  CH  MM
##        CH 157  63
##        MM   8  42
```

```r
#Error rate, calculated as misclassified/total
(63+8)/270
```

```
## [1] 0.262963
```

####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
tune_c <- tune(svm,
               Purchase ~ .,
               data = OJ.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 = 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:  1
##
## Number of Support Vectors:  325
##
##  ( 163 162 )
##
##
## 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, OJ.train)
Purchase4 <- predict(tuned_model, OJ.test)

#Confusion matrix for training set predictions
t1 <- table(predicted = Purchase3,
      true = OJ.train$Purchase)
```

```r
#Error rate, calculated as misclassified / total
(t1[1,2]+t1[2,1])/800
```

```
## [1] 0.1575
```

```r
#Confusion matrix for test set predictions
t2 <- table(predicted = Purchase4,
      true = OJ.test$Purchase)

#Error rate, calculated as misclassified / total
(t2[1,2]+t2[2,1])/270
```

```
## [1] 0.1814815
```

The optimal test error rates for training and test sets are 0.1575 and 0.1815, respectively for the tuned model. The test error rates for training and test sets are 0.2487 and 0.2629, respectively for the original model. This result shows that the tuned model has lower test error rates for all sets than the original model. This would have been possible since the tuning function determines the cost value which results in the most accurate classifier.