Intro to Machine Learning - Problem Set 3: Decision Trees and SVM

Ipek Cinar
2/16/2020

```
knitr::opts_chunk$set(echo = TRUE)
library(ggplot2)
library(tidyverse)
library(here)
library(ISLR)
library(broom)
library(rsample)
library(modelr)
library(rcfss)
library(yardstick)
library(MASS)
library(gbm)
library(rpart)
library(randomForest)
library(ipred)
```

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.

```
set.seed(1234)
biden_data <- read_csv("~/Desktop/Intro to Machine Learning/Problem Sets/PSet 3/nes2008.csv")
```

```
## Parsed with column specification:
## cols(
##   biden = col_double(),
##   female = col_double(),
##   age = col_double(),
##   educ = col_double(),
##   dem = col_double(),
##   rep = col_double()
## )
```

```
p <- biden_data %>% dplyr::select(-biden) %>% length()
lambda <- seq(0.0001, 0.04, by= 0.001)
```

2. 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(1234)
train_biden <- sample(1:nrow(biden_data), nrow(biden_data)*0.75)
test_biden <- setdiff(1:nrow(biden_data), train_biden)
```

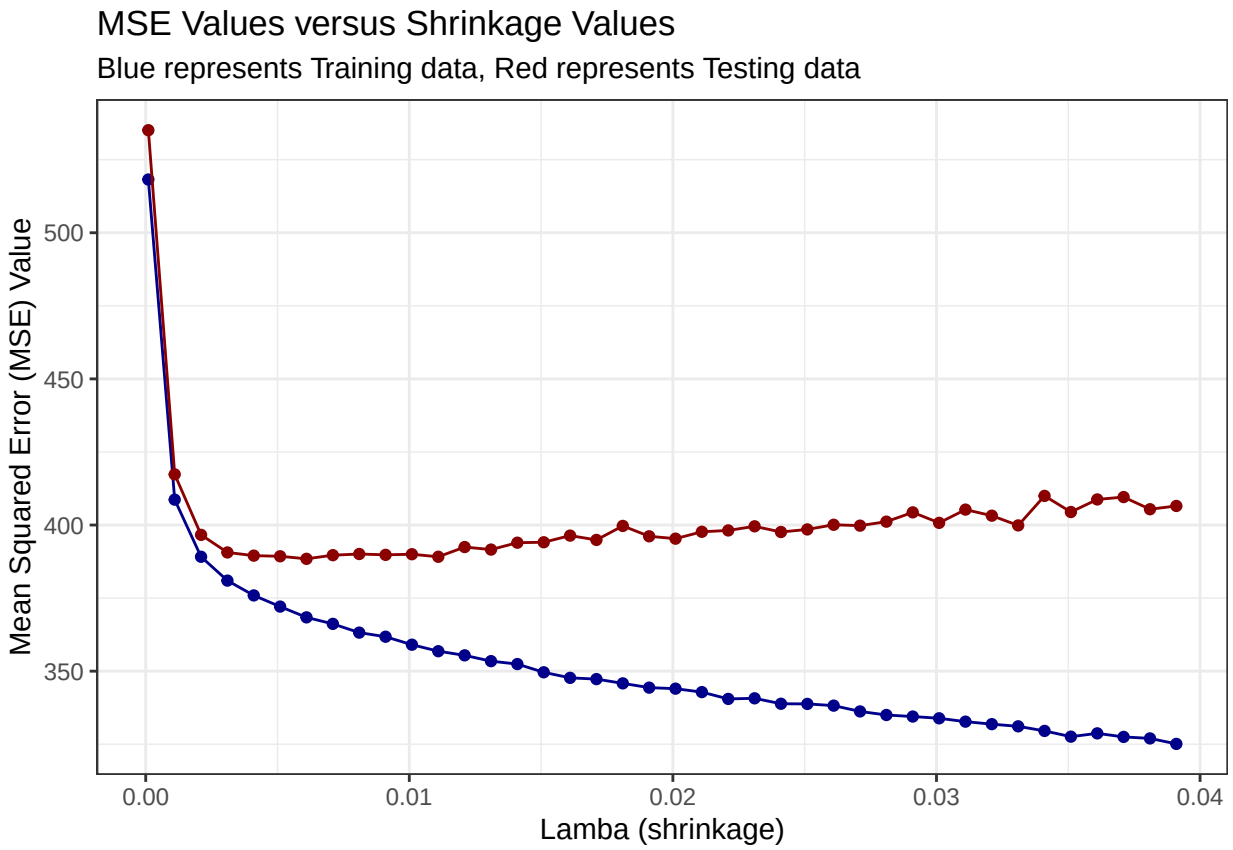
3. Create empty objects to store training and testing MSE, and then write a loop to perform boosting on the training set with 1,000 trees for the pre-defined range of values of the shrinkage parameter, λ. Then, plot the training set and test set MSE across shrinkage values.

```
# Writing the function to perform boosting on TRAIN/TEST set with 1000 trees
train_MSEs <- data.frame(matrix(ncol = 1, nrow = length(lambda))) %>% setNames("train_MSE")
test_MSEs <- data.frame(matrix(ncol = 1, nrow = length(lambda))) %>% setNames("test_MSE")

for (i in 1:length(lambda)){
  boosting_biden <- gbm(biden ~ .,
                        data = biden_data[train_biden, ],
                        distribution = "gaussian",
                        n.trees = 1000,
                        shrinkage = lambda[i],
                        #setting maximum depth of tree to 4
                        #as seen in class
                        interaction.depth = 4)
  train_prediction <- predict(boosting_biden, newdata = biden_data[train_biden, ],
                              n.trees = 1000)
  train_MSEs[i, ] = mean((biden_data$biden[train_biden] - train_prediction)^2)
  test_prediction <- predict(boosting_biden, newdata = biden_data[test_biden, ],
                             n.trees = 1000)
  test_MSEs[i, ] = mean((biden_data$biden[test_biden] - test_prediction)^2)
}

mse_train_values <- as.data.frame(cbind(lambda, train_MSEs))
mse_test_values <- as.data.frame(cbind(lambda, test_MSEs))

ggplot () + geom_point(data = mse_train_values,
                       mapping = aes(lambda, train_MSE), color = "darkblue") +
            geom_line(data = mse_train_values,
                      mapping = aes(lambda, train_MSE), color = "darkblue") +
            geom_point(data = mse_test_values,
                       mapping = aes(lambda, test_MSE), color = "darkred") +
            geom_line(data = mse_test_values,
                      mapping = aes(lambda, test_MSE), color = "darkred") +
            labs(title = "MSE Values versus Shrinkage Values",
                 subtitle = "Blue represents Training data, Red represents Testing data",
                 x = "Lamba (shrinkage)",
                 y = "Mean Squared Error (MSE) Value") + theme_bw()
```



4. 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?

```
  boosting_biden <- gbm(biden ~ .,
                        data = biden_data[train_biden, ],
                        distribution = "gaussian",
                        n.trees = 1000,
                        shrinkage = 0.01,
                        interaction.depth = 4)
  train_prediction <- predict(boosting_biden, newdata = biden_data[train_biden, ],
                              n.trees = 1000)
  train_MSE_fixed = mean((biden_data$biden[train_biden] - train_prediction)^2)
  test_prediction <- predict(boosting_biden, newdata = biden_data[test_biden, ],
                             n.trees = 1000)
  test_MSE_fixed = mean((biden_data$biden[test_biden] - test_prediction)^2)
# train_MSE_fixed
# test_MSE_fixed
```

The train MSE we get is 359.6047033 and the test MSE we get is 391.2987555 when we set λ equal to 0.01. As we can imply from the results, the test MSE is higher than the training MSE which can be considered as not a surprising outcome giben that the model was not trained on the test set. In addition, from the figure we have in question 3, we can see that as the shrinkage factor λ increases, the MSE of the training set decreases while the MSE of the test set is increasing, imply to as that an overfitting is taking place.

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

```
bagging_biden <- bagging(biden ~ .,
                         data = biden_data[train_biden, ])
testpredict_bagging <- predict(bagging_biden, newdata = biden_data[test_biden, ])
bagging_mse <- mean((biden_data$biden[test_biden] - testpredict_bagging)^2)
```

After applying bagging to the training set, the test set MSE we get is 398.3379712.

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

```
randomforest_biden <- randomForest(biden ~ .,
                                   data = biden_data[train_biden, ])
testpredict_randomforest <- predict(randomforest_biden, newdata = biden_data[test_biden, ])
randomforest_mse <- mean((biden_data$biden[test_biden] - testpredict_randomforest)^2)
```

After applying random forest to the training set, the test set MSE we get is 403.3312819.

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

```
lm_biden <- glm(biden ~ .,
                data = biden_data[train_biden, ])
testpredict_lm <- predict(lm_biden, newdata = biden_data[test_biden, ])
lm_mse <- mean((biden_data$biden[test_biden] - testpredict_lm)^2)
```

After applying random forest to the training set, the test set MSE we get is 391.1308614.

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

- Boosting(fixed shrinkage) MSE: 391.2987555
- Bagging MSE: 398.3379712
- Random Forest MSE: 403.3312819.
- LM MSE: 391.1308614

From the bullets above, we can see that the linear regression fit the best given the lower MSE value associated. This suggests that, given the results above, the other machine elarning techniques tended to overfit the data leading to worse performances on the test set/performing worse on the data that it has not seen. Hence, in this case, linear model would be more preferable given the comparison across the MSE values. This result holded true after I changed the seed a couple times (but its always possible that we get a different result with a different seed).

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(1234)
attach(OJ)
train_index_oj <- sample(1:nrow(OJ), 800)
train_oj <- OJ[train_index_oj, ]
test_oj <- OJ[-train_index_oj, ]
```

2. 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.

```
library(e1071)
library(caret)
svm_oj <- svm(Purchase ~ .,
              data = train_oj,
              kernel = "linear",
              cost = 0.01)
summary(svm_oj)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_oj, kernel = "linear", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  445
##
##  ( 223 222 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

As we can see from the summary table above, using a linear SVM kernal and a cost of 0.01, the number of support vectors we have for the training data is 445 with two classes almost equally split in betweent two: 223 vectors for one, 222 for the other). The levels are CH and MM which denotes the brand of the orange juice.

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

```
## Train set
train_prediction <- predict(svm_oj, newdata = train_oj)
train_statistics = confusionMatrix(train_prediction, train_oj$Purchase)
train_statistics$table
```

```
##           Reference
## Prediction  CH  MM
##         CH 432  76
##         MM  57 235
```

```
# train_statistics$overall[1] gives the accuracy,
# we can subtract it from 1 to get the error rate.
# Please disregard ""accuracy that appears along with it".
error_training <- 1 - train_statistics$overall[1]
error_training
```

```
## Accuracy
##  0.16625
```

```r
# Another way to calculate it is: (FN+FP)/N
error_training_otherway <- (76+57) / (432+76+57+235)
# error_training_otherway

## Test set
test_prediction <- predict(svm_oj, newdata = test_oj)
test_statistics = confusionMatrix(test_prediction, test_oj$Purchase)
test_statistics$table
```

```
##           Reference
## Prediction  CH  MM
##         CH 151  32
##         MM  13  74
```

```r
# test_statistics$overall[1] gives the accuracy,
# we can subtract it from 1 to get the error rate.
# Please disregard ""accuracy that appears along with it".
error_test <- 1 - test_statistics$overall[1]
error_test
```

```
##  Accuracy
## 0.1666667
```

```r
# Another way to calculate it is: (FN+FP)/N
error_test_otherway <- (32+13) / (151+74+32+13)
# error_test_otherway
```

The training set error rate is 0.16625 while the error for test set is 0.1666667.

4. 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_oj_tune <- tune(svm,
                    Purchase ~ .,
                    data = train_oj,
                    kernel = "linear",
                    ranges = list(cost = seq(0.1, 10, 0.1)))
# One way to get at the optimal cost:
optimal_1 <- svm_oj_tune$best.model
optimal_1
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train_oj,
##     ranges = list(cost = seq(0.1, 10, 0.1)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.2
##
## Number of Support Vectors:  347
```

```r
# Another way to get at the optimal cost:
optimal_2 <- svm_oj_tune$best.parameters
optimal_2
```

```
##   cost
```

```
## 2  0.2
```
As we can see from the above output, the optimal cost is 0.2.

5. 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.)

```
svm_oj_optimal <- svm(Purchase ~ .,
              data = train_oj,
              kernel = "linear",
              cost = 0.02)

train_prediction_tuned <- predict(svm_oj_optimal, newdata = train_oj)
train_tuned_statistics = confusionMatrix(train_prediction_tuned, train_oj$Purchase)
train_tuned_statistics$table
```

```
##           Reference
## Prediction  CH  MM
##         CH 429  75
##         MM  60 236
```

```
# train_statistics$overall[1] gives the accuracy,
# we can subtract it from 1 to get the error rate.
# Please disregard ""accuracy that appears along with it".
error_training_tuned <- 1 - train_tuned_statistics$overall[1]
error_training_tuned
```

```
## Accuracy
##  0.16875
```

```
# Another way to calculate it is: (FN+FP)/N
error_training_tuned_otherway <- (75+60) / (429+75+60+236)
# error_training_otherway

## Test set
test_prediction_tuned <- predict(svm_oj_optimal, newdata = test_oj)
test_tuned_statistics = confusionMatrix(test_prediction_tuned, test_oj$Purchase)
test_tuned_statistics$table
```

```
##           Reference
## Prediction  CH  MM
##         CH 151  30
##         MM  13  76
```

```
# test_statistics$overall[1] gives the accuracy,
# we can subtract it from 1 to get the error rate.
# Please disregard ""accuracy that appears along with it".
error_test_tuned <- 1 - test_tuned_statistics$overall[1]
error_test_tuned
```

```
##  Accuracy
## 0.1592593
```

```
# Another way to calculate it is: (FN+FP)/N
error_test_tuned_otherway <- (30+13) / (151+30+13+76)
# error_test_otherway
```

The new error rate for training set is 0.16875 while the new error rate for test is 0.1592593 after using the optimal cost value we found above. Compared to the previous value of error rate for the training set (which

was 0.16625), the error rate is about -0.0025 worse (error rate is very minimally higher when we use the optimal cost), while for test set it is 0.0074074 better. Looking at the error rate comparison, we can see that the error rate is sligtly better when we use the optimal cost in the test set (with data it has not seen before). Even though the change is very minimal in substantive terms, we can say that using the optimal cost gave us a slighly lower error rate and hence better accuracy.