PS3

Bohan Yin

2/16/2020

Decision Trees

```r
set.seed(1234)
data_biden <- read.csv("./data/nes2008.csv")
p <- length(colnames(data_biden))-1
lambda <- seq(0.0001, 0.04, 0.001)
```

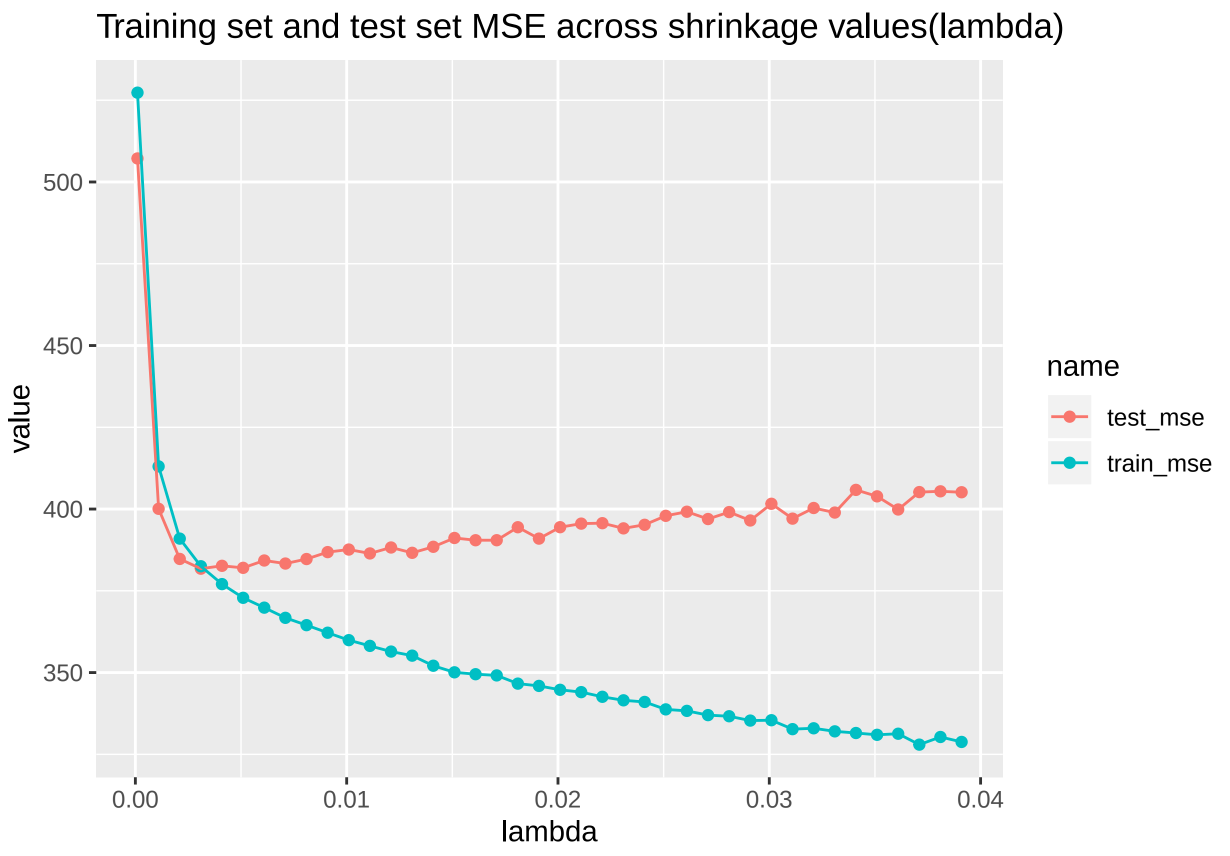
2. (10 points) Create a training set consisting of 75% of the observations, and a test set with all remaining obs. Note: because you will be asked to loop over multiple λ values below, these training and test sets should only be integer values corresponding with row IDs in the data. This is a little 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
set.seed(1234)
trainIndex <- createDataPartition(data_biden$biden,p=0.75,list=FALSE)
train_biden <- data_biden[trainIndex,]
test_biden <- data_biden[-trainIndex,]
```

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
set.seed(1234)
# write a mse function
get_mse <- function(lambda, new_data) {
  boost <- gbm(biden ~ .,
                    data=train_biden,
                    distribution="gaussian",
                    n.trees=1000,
                    shrinkage=lambda,
                    interaction.depth = 4)
  preds = predict(boost, newdata=new_data,
                  n.trees = 1000)
  mse = mean((preds - new_data$biden)^2)
  return(mse)
}

# create a table storing mse values
mse_tbl <- data.frame(lambda = double(), mse_train = double(), mse_test = double())
for (i in lambda) {
  train_mse = get_mse(i, train_biden)
  test_mse = get_mse(i, test_biden)
  mse_tbl = rbind(mse_tbl, c(i, train_mse, test_mse))
}
mse_tbl <- mse_tbl %>%
  rename(
  `lambda` = X1e.04,
  `train_mse` = X527.308490568936,
  `test_mse` =X507.19824146991)
# plotting
mse_tbl %>%
  pivot_longer(c(test_mse, train_mse)) %>%
  ggplot(aes(lambda, value, color = name)) +
  geom_point() +
  geom_line() +
  labs(
    title = "Training set and test set MSE across shrinkage values(lambda)"
  )
```



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?

```
set.seed(1234)
tibble(Lambda = 0.01, Train_mse = get_mse(0.01, train_biden), Test_mse = get_mse(0.01, test_biden))
```

```
## # A tibble: 1 x 3
##   Lambda Train_mse Test_mse
##    <dbl>     <dbl>    <dbl>
## 1   0.01      361.     385.
```

The tested mse is 385.2511877, which is higher than the trained mse 359.8900272, meaning that the decesion tree model performs better under the training set. Also from the plot we can see that this test mse is close to the optimal MSE value.

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

```
library(ipred)
set.seed(1234)

# train bagged model
biden_bag1 <- bagging(
  formula = biden ~ .,
  data = train_biden,
  nbagg = 1000,
  coob = TRUE,
)
pred_bag <- predict(biden_bag1, newdata = test_biden)
mse_bag <- (test_biden$biden - pred_bag)^2
paste("The test set MSE for bagging is", mean(mse_bag))
```

```
## [1] "The test set MSE for bagging is 379.042219738577"
```

The test set MSE for bagging approach is 379.0422197.

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

```
set.seed(1234)
biden_rf <- randomForest(biden ~ .,
                   data = train_biden)
pred_rf <- predict(biden_rf, newdata = test_biden)
mse_rf <- (test_biden$biden - pred_rf)^2
paste("The test set MSE for random forest is", mean(mse_rf))
```

```
## [1] "The test set MSE for random forest is 390.052714568253"
```

The test set MSE for random forest approach is 390.0527146

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

```r
set.seed(1234)
biden_lm <- lm(biden ~.,
               data = train_biden)
pred_lm <- predict(biden_lm, newdata = test_biden)
mse_lm <- (test_biden$biden - pred_lm)^2
paste("The test set MSE for linear regression is", mean(mse_lm))
```

```
## [1] "The test set MSE for linear regression is 373.766333399188"
```

The test set MSE for linear regression approach is 373.7663334

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

After comparing test errors across all fits, it seems that linear regression approach performs the best result, since it has the lowest test set MSE of 373.7663334. This might indicate that decision tree models tend to have the problem of overfitting, leading to less ideal performance on new datasets.

Support Vector Machines

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

```r
oj_data <- ISLR::OJ
set.seed(1234)
tr <- sample(1:nrow(OJ), 800)
oj_train <- OJ[tr,]
oj_test <- OJ[-tr,]
```

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
set.seed(1234)
svmfit <- svm(Purchase ~ .,
             data = oj_train,
             kernel = "linear",
             cost = 0.01); summary(svmfit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = oj_train, kernel = "linear",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  437
##
##  ( 219 218 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
pred_svm <- predict(svmfit, newdata = oj_test)
knitr::kable(tibble(test_accuracy = mean((pred_svm == oj_test$Purchase))))
```

test_accuracy
0.8407407

From the result we can see there are 437 support vectors, and 2 classes. The accuracy for this approach is 0.8407407, which is not bad in general. However, considering that the sample size is 800, 437 support vectors seem too many.

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

```r
set.seed(2345)
table(pred_svm, oj_test$Purchase)
```

```
##
## pred_svm  CH  MM
##       CH 151  24
##       MM  19  76
```

```r
test_error <- 1-mean(pred_svm == oj_test$Purchase)
train_error <- 1-mean(svmfit$fitted == oj_train$Purchase)
paste("The test error rate is", test_error)
```

```
## [1] "The test error rate is 0.159259259259259"
```

```r
paste("The train error rate is", train_error)
```

```
## [1] "The train error rate is 0.16875"
```

The training set error rate is 0.16875, and test set error rate is 0.1592593. The test error rate is lower than train error rate.

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).

```
set.seed(2345)

tune_c <- tune(svm,
                Purchase ~ .,
                data = oj_train,
                kernel = "linear",
                ranges = list(cost = seq(0.01, 10, by = 0.1)))
tune_c$best.parameters
```

```
##   cost
## 4 0.31
```

```
tuned_model <- tune_c$best.model
```

From the result we see that the optimal cost is 0.31.

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.)

```
# Test Error
op_test_error <- 1- mean(predict(tuned_model, newdata = oj_test) == oj_test$Purchase)
# Train Error
op_train_error <- 1- mean(predict(tuned_model, newdata = oj_train) == oj_train$Purchase)
paste("The optimal test error rate is", op_test_error)
```

```
## [1] "The optimal test error rate is 0.155555555555556"
```

```
paste("The optimal train error rate is", op_train_error)
```

```
## [1] "The optimal train error rate is 0.165"
```

The optimal training error rate under the cost of 1 is 0.165, and the optimal testing error rate under the cost of 1 is 0.1555556. The confusion matrix for testing MSE is:

```
table(predict(tuned_model, oj_test), oj_test$Purchase)
```

```
##
##       CH  MM
##   CH 150  22
##   MM  20  78
```

The confusion matrix for training MSE is:

```r
table(predict(tuned_model, oj_train), oj_train$Purchase)
```

```
##
##       CH  MM
##   CH 422  71
##   MM  61 246
```

The confusion matrix from tuned classifier and original classifier do not differ significantly from each other. In sum, both optimal testing error and training error are lower compared to the error rates before tuning. Alghough the differences are within 0.004, which could not indicate a significant difference, the optimally tuned classifier performs better, and has slightly higher accuracy rate. This could be explained by the fact that given a less strict cost, the model has a relatively higher tolerance on classifying observations, so the variance decreases, reducing the error rate.