Intro to ML Hwk 3

William Ansehl

2/17/2020

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

Decision Trees

Problem 1

```r
set.seed(222)
nesdata <- read.csv('nes2008.csv')
p <- names(nesdata[c(-1)])
lambdas = seq(from=0.0001,to=0.04, by=0.001)
```

Problem 2

```r
population_size <- nrow(nesdata)
sample_size <- 0.75*population_size
indices <- sample(seq_len(population_size), sample_size)
train <- nesdata[indices,]
test <- nesdata[-indices,]
```

Problem 3

```r
train_list <- list()
test_list <- list()

library(gbm)
```

```
## Loaded gbm 2.1.5
```

```r
for (value in lambdas) {
  boost.nesdata <- gbm(biden ~ .,
                    data=train,
                    distribution="gaussian",
```

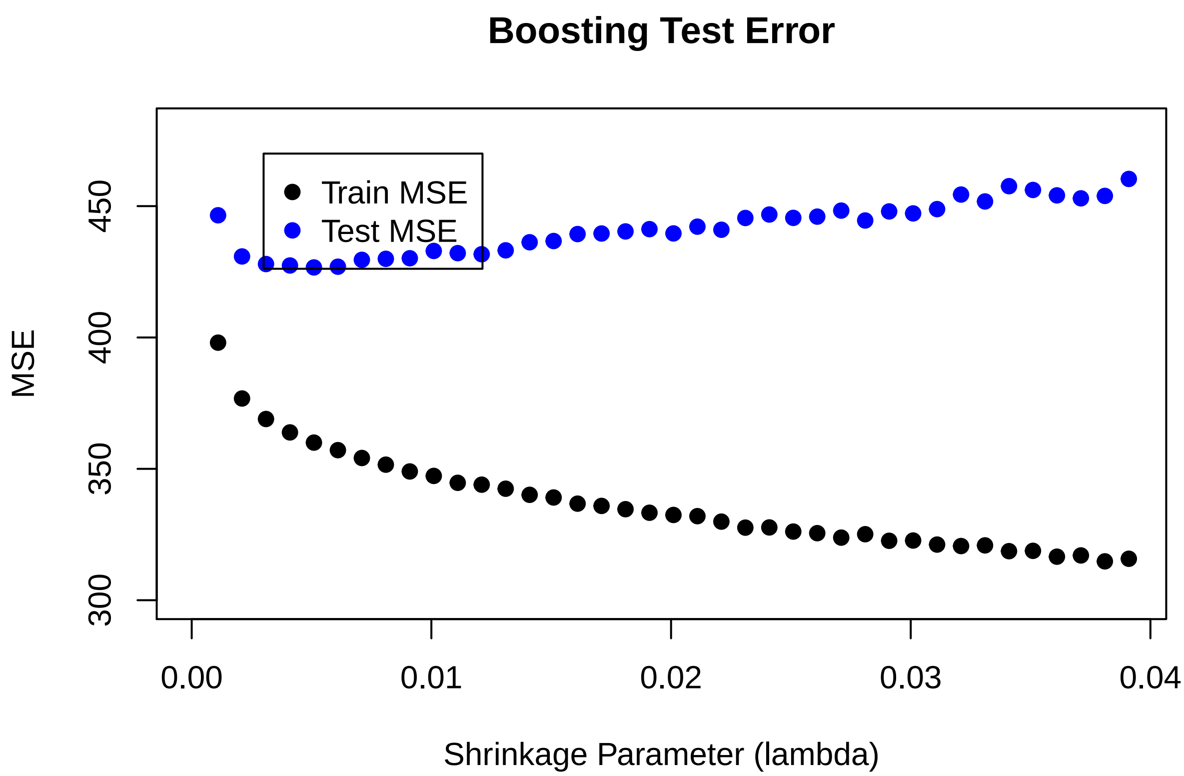
```
                    n.trees=1000,
                    shrinkage=value,
                    interaction.depth = 4)

  predsTrain = predict(boost.nesdata, newdata=train,n.trees = 1000)
  predsTest = predict(boost.nesdata, newdata=test,n.trees = 1000)

  mseTrain = mean((predsTrain-train$biden)^2)
  mseTest = mean((predsTest-test$biden)^2)

  train_list <- c(train_list, mseTrain)
  test_list <- c(test_list, mseTest)
}

plot(lambdas, train_list, pch = 19, col = 'black',ylab = "MSE", xlab = "Shrinkage Parameter (lambda)", m
points(lambdas, test_list, pch = 19, col = "blue")
legend(0.003, 470,legend=c("Train MSE", "Test MSE"),
       col=c("black", "blue"),pch=19)
```



Problem 4

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

```r
preds = predict(boost.nesdata, newdata=test,n.trees = 1000)
mse <- mean((preds-test$biden)^2)
mse
```

```
## [1] 430.6261
```

```r
# mse = 430.6261
min(unlist(test_list))
```

```
## [1] 426.6643
```

```r
# mse = 426.6643

# The test set mse at the lambda = 0.01 level is 430.6261.
# This value is a little bit higher than the miminum mse from the
# test list when we evaluated mse at different shrinkage parameters.
# I predict that the optimal shrinkage parameter is around 0.003-0.004
```

Problem 5

```r
library('ipred')
bag <- bagging(
  formula = biden ~ .,
  data = nesdata,
  nbagg = 100,
  coob = TRUE
)
predsBag <- predict(bag, newdata = test)
mseBag <- mean((predsBag - test$biden)^2)
mseBag
```

```
## [1] 429.5398
```

```r
# mseBag = 429.5398
# The bagged model performs a little better than the boosted model
# with a shrinkage parameter of 0.01.
# However, the bagged model still does not perform as well as the
# optimized boosted model with a shrinkage parameter at around 0.003-0.004
```

Problem 6

```r
library('randomForest')
```

```
## randomForest 4.6-14

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

```r
randForest <- randomForest(biden ~., data = train)
predsRF <- predict(randForest, newdata = test)
mseRF <- mean((predsRF - test$biden)^2)
mseRF
```

```
## [1] 436.5406
```

```r
# mseRF = 436.5406
# The random forest model performs worse than the boosted model at its
# optimal shrinkage parameter as well as the bagged model
```

Problem 7

```r
linearModel <- lm(biden~., data = train)
predsLM <- predict(linearModel, newdata = test)
mseLM <- mean((predsLM - test$biden)^2)
mseLM
```

```
## [1] 424.706
```

```r
# mseLM = 424.706
# The linear model thus far is the best performing model in that it has
# the lowest mse thus far
```

Problem 8

```r
table <- matrix(c(426.6643, 429.5398, 436.5406, 424.706), ncol = 1, byrow=TRUE)
rownames(table) <- c('Boosting (Optimal)', 'Bagging', 'Random Forest', 'Linear Regression')
colnames(table) <- c('MSE')
table <- as.table(table)
table
```

```
##                          MSE
## Boosting (Optimal) 426.6643
## Bagging            429.5398
## Random Forest      436.5406
## Linear Regression  424.7060
```

```r
#From the MSEs of the generated models, we discovered that
# Linear Regression performed the best, followed by boosting,
# bagging and last random forests. Therefore, linear regression
# is seemingly the best fit for the data. However, boosting, bagging
# and linear regression performed relatively similar with random
# forests not far behind.
```

SVM

Problem 1

```r
library('ISLR')
ojdata <- OJ
population_size <- nrow(ojdata)
sample_size <- 800
indices <- sample(seq_len(population_size), sample_size)
train <- ojdata[indices,]
test <- ojdata[-indices,]
```

Problem 2

```r
library('e1071')
svmfit <- svm(Purchase ~ .,
              data = train,
              kernel = "linear",
              type = 'C-classification',
              cost = 0.01,
              scale = FALSE); summary(svmfit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "linear", type = "C-classification",
##     cost = 0.01, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  623
##
##  ( 311 312 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
# From the summary, we see that the svm classifier used
# 623 total classifiers. Of this total, 312 belonged to
# Citrus Hill and 311 belonged to Minute Maid. The cause
# behind the high quantitity of support vectors is probably
# due to our low cost value, which is 0.01. In addition,
# there are only two classes because we used 'Purchase' as
# the response variable. Since 'Purchase' takes on only two
# values - CH or MM - then there can only be two classes.

# It is visually difficult to understand svms. For example,
# a support vector classifier for 1-D data is a point. A
# support vector classifier for 2-D data is a line. A support
# vector classifier for 3-D data is a plane and so on. Since
# our data has many features and is thus represented in higher
# dimensions, the support vector classifier is a hyperplane.
```

Problem 3

```r
predTrain = predict(svmfit, train)
predTest = predict(svmfit, test)
set.seed(222)
```

```r
table(predicted = predTrain, actual = train$Purchase)
```

```
##          actual
## predicted  CH  MM
##        CH 431 113
##        MM  51 205
```

```r
# 437 + 183 = 620 correctly classified and 187 incorrectly classified
# training set error rate = 180/800 * 100 = 22.5%

table(predicted=predTest, actual = test$Purchase)
```

```
##          actual
## predicted  CH  MM
##        CH 152  45
##        MM  19  54
```

```r
# 148 + 67 = 215 correctly classified out of 270
# test set error rate = 55/270 * 100 = 20.37%
```

Problem 4

```r
tune_c <- tune(svm,
               Purchase ~ .,
               data = train,
               kernel = "linear",
               ranges = list(cost = c(0.01, 0.1, 1, 10, 100, 250, 500, 750, 1000)
               ))
summary(tune_c)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.165
##
## - Detailed performance results:
##      cost   error dispersion
## 1    0.01 0.17250 0.03162278
## 2    0.10 0.16500 0.03525699
## 3    1.00 0.16750 0.04090979
## 4   10.00 0.16875 0.03875224
## 5  100.00 0.16625 0.04168749
## 6  250.00 0.16625 0.04168749
## 7  500.00 0.16625 0.04168749
## 8  750.00 0.16625 0.04168749
## 9 1000.00 0.16750 0.04297932
```

```r
modelTuned <- tune_c$best.model
summary(modelTuned)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train, ranges = list(cost = c(0.01,
##     0.1, 1, 10, 100, 250, 500, 750, 1000)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##
## Number of Support Vectors:  344
##
##  ( 174 170 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
# The optimal cost value is 0.1 which includes  a classifier with 334
# total support vectors - 166 belonging to CH and 168 belonging to MM.
```

Problem 5

```r
tune_c <- tune(svm,
               Purchase ~ .,
               data = train,
               kernel = "linear",
               ranges = list(cost = c(0.01, 0.1, 1, 10, 100, 250, 500, 750, 1000)
               ))
summary(tune_c)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.165
##
## - Detailed performance results:
##      cost   error dispersion
## 1    0.01 0.16500 0.03162278
## 2    0.10 0.17125 0.03821086
## 3    1.00 0.16625 0.03821086
```

```
## 4   10.00 0.17000 0.02838231
## 5  100.00 0.16875 0.02517301
## 6  250.00 0.16750 0.02838231
## 7  500.00 0.16750 0.02838231
## 8  750.00 0.16750 0.02958040
## 9 1000.00 0.16750 0.03291403
```

```
modelTuned <- tune_c$best.model
summary(modelTuned)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train, ranges = list(cost = c(0.01,
##     0.1, 1, 10, 100, 250, 500, 750, 1000)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  435
##
##  ( 217 218 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
# The optimal cost value is 0.1 which includes  a classifier with 332
# total support vectors - 165 belonging to CH and 167 belonging to MM.

## Problem 5
predTrain = predict(modelTuned, train)
predTest = predict(modelTuned, test)
table(predicted = predTrain, actual = train$Purchase)
```

```
##          actual
## predicted  CH  MM
##        CH 427  76
##        MM  55 242
```

```
# With the optimal cost value of 0.1, 430 + 240 = 670 correctly predicted of 800.
# 138 incorrectly predicted --> 130/800 * 100 = 16.25% error rate in the training set

table(predicted = predTest, actual = test$Purchase)
```

```
##          actual
## predicted  CH  MM
##        CH 150  30
##        MM  21  69
```

```
# With the optimal cost value of 0.1, 143 + 86 = 229 of 270 total correctly predicted
# 38 incorrectly predicted --> 41/270 * 100 = 15.19% error rate in the testing set
```

```
# When using the optimal cost value of 0.1 in the tuned model, we correctly classified
# approximately 6% more of the data. Similarly, we correctly classified approximately
# 5% more of the test set data.
# In addition, in the svm model, many of the incorrect predictions were Minute Maids
# we wrongly classified as Citrus Hill. The tuned model experienced particularly fewer
# mis-classifications in this area and as a result led to a better-fit model with better
# predictive power.
```