PS3_Jiewen Luo

Jiewen Luo

2/16/2020

```r
library(tree)

## Warning: package 'tree' was built under R version 3.6.2

library(ISLR)

## Warning: package 'ISLR' was built under R version 3.6.2

library(tidyverse)

## Warning: package 'tidyverse' was built under R version 3.6.2

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

## -- Attaching packages ------------------------------------------------ tid
yverse 1.3.0 --

## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## Warning: package 'ggplot2' was built under R version 3.6.2

## Warning: package 'tibble' was built under R version 3.6.2

## Warning: package 'tidyr' was built under R version 3.6.2

## Warning: package 'readr' was built under R version 3.6.2

## Warning: package 'purrr' was built under R version 3.6.2

## Warning: package 'dplyr' was built under R version 3.6.2

## Warning: package 'stringr' was built under R version 3.6.2

## Warning: package 'forcats' was built under R version 3.6.2

## -- Conflicts --------------------------------------------------- tidyverse
_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

library(broom)
```

```
## Warning: package 'broom' was built under R version 3.6.2

library(rsample)

## Warning: package 'rsample' was built under R version 3.6.2

library(rcfss)
library(yardstick)

## Warning: package 'yardstick' was built under R version 3.6.2

## For binary classification, the first factor level is assumed to be the eve
nt.
## Set the global option `yardstick.event_first` to `FALSE` to change this.

##
## Attaching package: 'yardstick'

## The following object is masked from 'package:readr':
##
##     spec
```

Decision Trees

1.

```
set.seed(23478)
da<-read.csv(file.choose())
da<-as_tibble(da)
p <- da[,-1]
lambda<-seq(from=0.0001, to=0.04,0.001 )
```

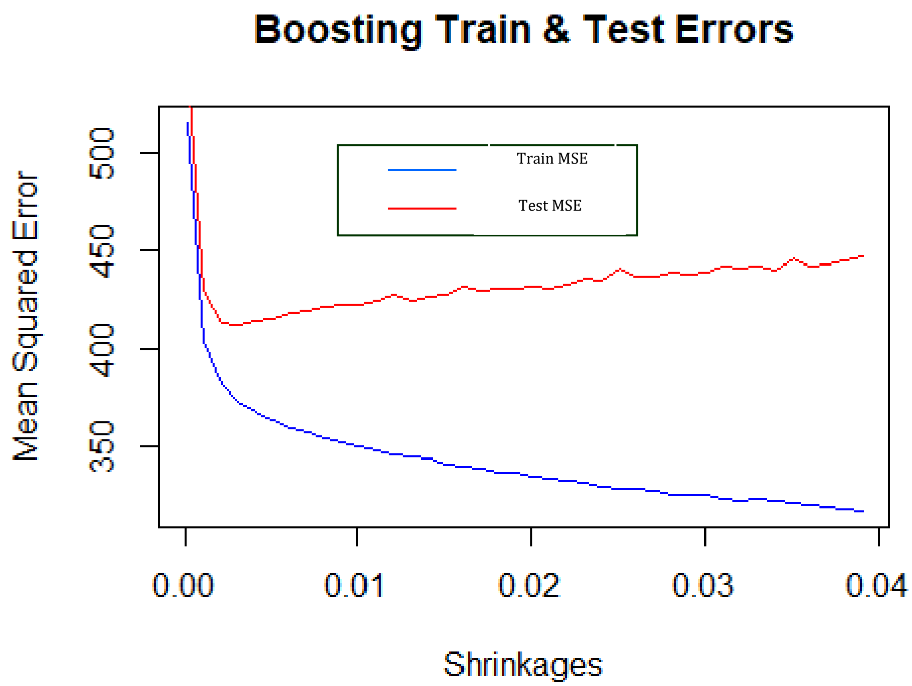
2.

```
drop_na(da)

## # A tibble: 1,807 x 6
##    biden female   age  educ   dem   rep
##    <int>  <int> <int> <int> <int> <int>
##  1    90      0    19    12     1     0
##  2    70      1    51    14     1     0
##  3    60      0    27    14     0     0
##  4    50      1    43    14     1     0
##  5    60      1    38    14     0     1
##  6    85      1    27    16     1     0
##  7    60      1    28    12     0     0
##  8    50      0    31    15     1     0
##  9    50      1    32    13     0     0
## 10    70      0    51    14     1     0
## # ... with 1,797 more rows
```

```
train<-sample(1:nrow(da),0.75*nrow(da))
training<-da[train,]
testing<-da[-train,]
```

3.

```
library(gbm)

## Warning: package 'gbm' was built under R version 3.6.2

## Loaded gbm 2.1.5

train_mse<- tibble(lambda, MSE=0)
test_mse<-tibble(lambda, MSE=0)
dim(train_mse)

## [1] 40  2

dim(test_mse)

## [1] 40  2

i=0
for (la in lambda){
  i=i+1
  boost<-gbm(biden ~ .,
  data=training,
  distribution="gaussian",
  n.trees=1000,
  shrinkage=la,
  interaction.depth = 4)
  preds_train = predict(boost, newdata=training,n.trees = 1000)
  preds_test = predict(boost, newdata=testing,  n.trees = 1000)
  train_mse[i,2]<-mean((preds_train-training$biden)^2)
  test_mse[i,2]=mean((preds_test-testing$biden)^2)
}

plot(lambda, train_mse$MSE ,
     pch=19,type="l",
     col="blue",
     ylab="Mean Squared Error",
     xlab="Shrinkages",
     main="Boosting Train & Test Errors")
lines(lambda,test_mse$MSE, col="red")
legend(1,8,legend = c("train_mse","test_mse"), col=c("blue","red"), lty=1:2,
cex=0.8)
```



4.

```
boost2<-gbm(biden ~ .,
  data=training,
  distribution="gaussian",
  n.trees=1000,
  shrinkage=0.01,
  interaction.depth = 4)
  preds_train2 = predict(boost2, newdata=training,  n.trees = 1000)
  preds_test2 = predict(boost2, newdata=testing,  n.trees = 1000)
  cat("Train mse when lamda equals to 0.01:", train_mse2<-mean((preds_train2-
training$biden)^2))

## Train mse when lamda equals to 0.01: 350.266

  cat("
         ")

  cat("Test mse when lamda equals to 0.01:", test_mse2=mean((preds_test2-test
ing$biden)^2))

## Test mse when lamda equals to 0.01: 424.1279
```

By comparing this test MSE with other test MSEs shown in the graph above, we can tell that the test MSE is pretty small when we pick lambda = 0.01. The test MSE is only slightly higher than the train MSE, suggesting that the variance is also low. Therefore, setting λ equal to 0.01 is a pretty reasonable choice.

5. Bagging

```
library(caret)

## Warning: package 'caret' was built under R version 3.6.2

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following objects are masked from 'package:yardstick':
##
##     precision, recall

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

library(Ecdat)

## Warning: package 'Ecdat' was built under R version 3.6.2

## Loading required package: Ecfun

## Warning: package 'Ecfun' was built under R version 3.6.2

##
## Attaching package: 'Ecfun'

## The following object is masked from 'package:base':
##
##     sign

##
## Attaching package: 'Ecdat'

## The following object is masked from 'package:datasets':
##
##     Orange

library(ipred)

## Warning: package 'ipred' was built under R version 3.6.2

library(vcd)

## Warning: package 'vcd' was built under R version 3.6.2

## Loading required package: grid

##
## Attaching package: 'vcd'
```

```
## The following object is masked from 'package:ISLR':
##
##     Hitters
```

```r
set.seed(886)
bag<-bagging(biden~., data=training, nbagg=100, coob=TRUE)
pred_bag<-predict(bag, newdata=testing)

print(bag_mse<-mean((pred_bag-testing$biden)^2))
```

```
## [1] 408.1462
```

6.Random Forest

```r
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.6.2

## randomForest 4.6-14

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

##
## Attaching package: 'randomForest'

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

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

```r
rf=randomForest(biden~., data = training)
pred_rf<-predict(rf, newdata=testing)

print(rf_mse<-mean((pred_rf-testing$biden)^2))
```

```
## [1] 420.1418
```

7. Linear Model

```r
library(tidyr)
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 3.6.2

##
## Attaching package: 'MASS'

## The following object is masked from 'package:Ecdat':
##
##     SP500
```

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

```
lm<-glm(biden~., data = training)
pred_lm<-predict(lm, newdata=testing)
print(lm_mse<-mean((pred_lm-testing$biden)^2))
```

```
## [1] 405.4105
```

8

For all the MSEs obtained from the above methods, linear regression is the best in the sense that it yields the smallest MSE. The MSEs from the 3 ensemble approaches are all slightly larger but do not have significant variation.

2. SVM

```
library(ISLR)
dt<-as_tibble(OJ)
```

1.

```
set.seed(1996)
sampling<-sample(1:nrow(dt), 800)
training<-dt[sampling, ] #traning set
testing<-dt[-sampling, ] #test set
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.6.2
```

2.

```
P_svm <- svm(as.factor(Purchase) ~ .,
             data = training,
             kernel = "linear",
             cost = 0.01,
             scale = FALSE)
summary(P_svm)
```

```
##
## Call:
## svm(formula = as.factor(Purchase) ~ ., data = training, kernel = "linear",

##     cost = 0.01, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
```

```
##
## Number of Support Vectors:  624
##
##  ( 313 311 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

There were 624 support vectors, 313 in class “CH” and 311 in the class “MM”. .\

3.

```
testing$Purchase<-as.factor(testing$Purchase)
training$Purchase<-as.factor(training$Purchase)

train_result<-predict(P_svm, training)
table(predicted_train=train_result, true = training$Purchase)

##                true
## predicted_train  CH  MM
##              CH 423 120
##              MM  59 198

Psvm_pred <- predict(P_svm, testing)
table(predicted_test=Psvm_pred, true = testing$Purchase)

##               true
## predicted_test  CH  MM
##             CH 146  37
##             MM  25  62

Error_train<-length(which(train_result!=training$Purchase))/length(training$P
urchase)
cat('Train Set Error Rate:', Error_train)

## Train Set Error Rate: 0.22375

cat("                 ")

Error_test<-length(which(Psvm_pred!=testing$Purchase))/length(testing$Purchas
e);
cat('Test set Error Rate:', Error_test)

## Test set Error Rate: 0.2296296
```

Based on our SVM, around 22.38% of the train observations are wrongfully classified, and around 22.96%% of the test observations are wrongfully classified. The error rates for the train data and test data are low and consistent, which means this classifier is quite a good fit.

4.

```
set.seed(202)

tune_svm <- tune(svm, as.factor(Purchase) ~ ., data = training, kernel = "lin
ear",
ranges = list(cost = c(0.01, 0.1, 1, 10, 100, 1000)))

# CV errors
summary(tune_svm)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.1775
##
## - Detailed performance results:
##    cost   error dispersion
## 1 1e-02 0.18000 0.03496029
## 2 1e-01 0.17875 0.03998698
## 3 1e+00 0.17875 0.04084609
## 4 1e+01 0.17750 0.04281744
## 5 1e+02 0.18125 0.04299952
## 6 1e+03 0.18250 0.03184162

# best:
tuned_model <- tune_svm$best.model
summary(tuned_model)

##
## Call:
## best.tune(method = svm, train.x = as.factor(Purchase) ~ ., data = training
,
##     ranges = list(cost = c(0.01, 0.1, 1, 10, 100, 1000)), kernel = "linear
")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
```

```
##
## Number of Support Vectors:  328
##
##  ( 165 163 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Based on the reported results we can tell that the best classifier has cost=10

5.

```
# predict Purchase labels

Btrain_pred <- predict(tuned_model, training)
table(B_train_pred = Btrain_pred, true = training$Purchase)

##             true
## B_train_pred  CH  MM
##           CH 424  77
##           MM  58 241

Btest_pred <- predict(tuned_model, testing)
table(B_test_pred = Btest_pred, true = testing$Purchase)

##            true
## B_test_pred  CH  MM
##          CH 154  26
##          MM  17  73

#Error Rate
Error_Btrain<-length(which(Btrain_pred!=training$Purchase))/length(training$P
urchase);
cat('Optimal Training set Error Rate:',Error_Btrain)

## Optimal Training set Error Rate: 0.16875

Error_Btest<-length(which(Btest_pred!=testing$Purchase))/length(testing$Purch
ase);
cat('Optimal Test set Error Rate:', Error_Btest)

## Optimal Test set Error Rate: 0.1592593
```

Based on our result, the optimal training set error rate is around 16.88%, and the optimal test set error rate is around 15.93%. By comparing these errors to the errors from question 3, we can tell that the optimal train error and test error are significantly lower. Therefore, even though using the “general rule of thumb” cost of 0.01 doesn’t seem like a bad choice, we did find evidence support model tuning.