MLPS3_YingZhou

Ying Zhou

2020/2/14

##Decision Trees ##1

```
Warning=FALSE
library(tidyverse)
```

```
## -- Attaching packages --------------------------- tidyverse 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   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

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

```
library(modelr)
library(tree)
```

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

```
library(randomForest)
```

```
## 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
library(ISLR)
library(patchwork)
library(rcfss)
```

```
##
## Attaching package: 'rcfss'

## The following object is masked from 'package:modelr':
##
##     mse
```

```r
library(gbm)
```

```
## Loaded gbm 2.1.5
```

```r
library(gganimate)
```

```
## No renderer backend detected. gganimate will default to writing frames to separate files
## Consider installing:
## - the `gifski` package for gif output
## - the `av` package for video output
## and restarting the R session
```

```r
library(rpart)
library(rpart.plot)
library(ggdendro)
library(broom)
```

```
##
## Attaching package: 'broom'

## The following object is masked from 'package:modelr':
##
##     bootstrap
```

```r
library(rsample)
library(yardstick)
```

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

##
## Attaching package: 'yardstick'

## The following objects are masked from 'package:modelr':
##
##     mae, mape, rmse

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

```r
library(randomForest)
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:patchwork':
##
##     area

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

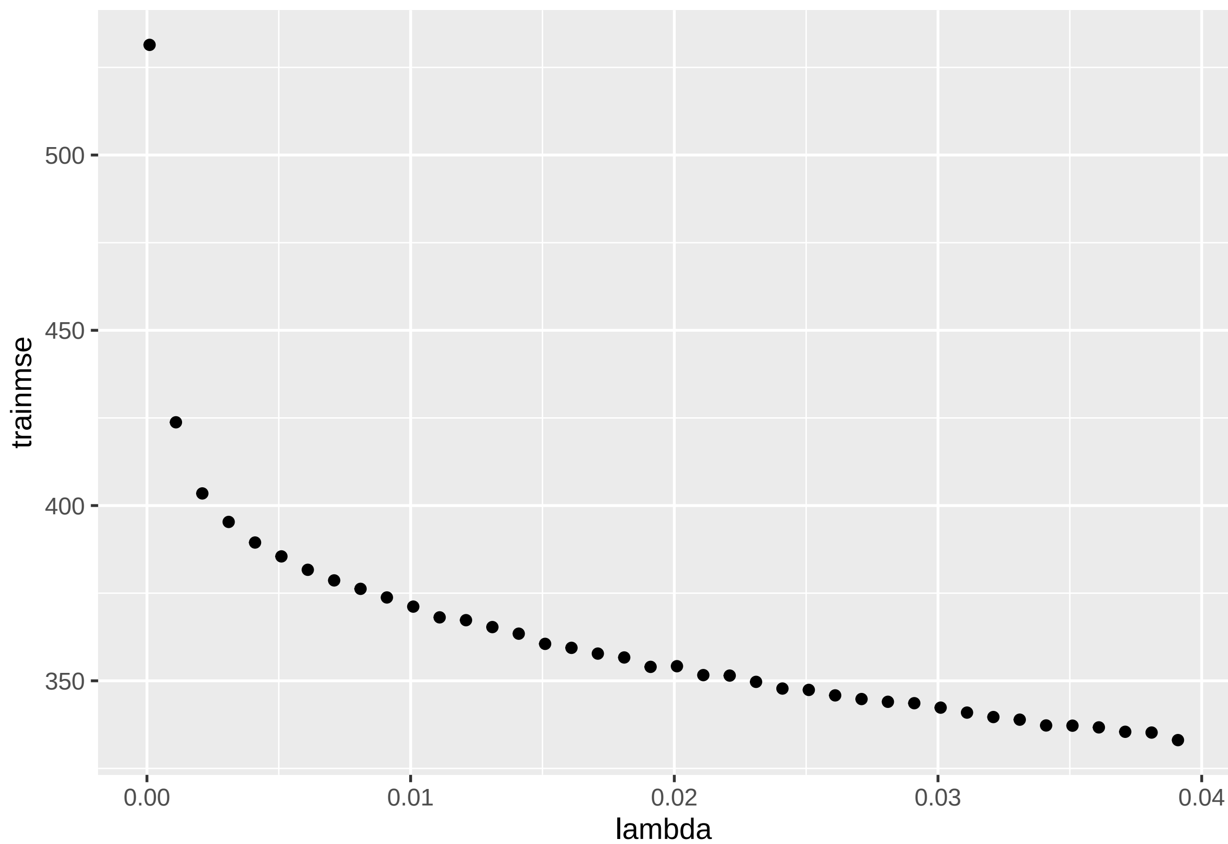
```r
library(ipred)
set.seed(1234)
nesdata <- read.csv("C:/Users/zhouy/Desktop/Uchicago/uchicourse/Intro to Machine Learning/PS/PS3/nes2008

lambda<-seq(0.0001,0.04,0.001)
```

##2

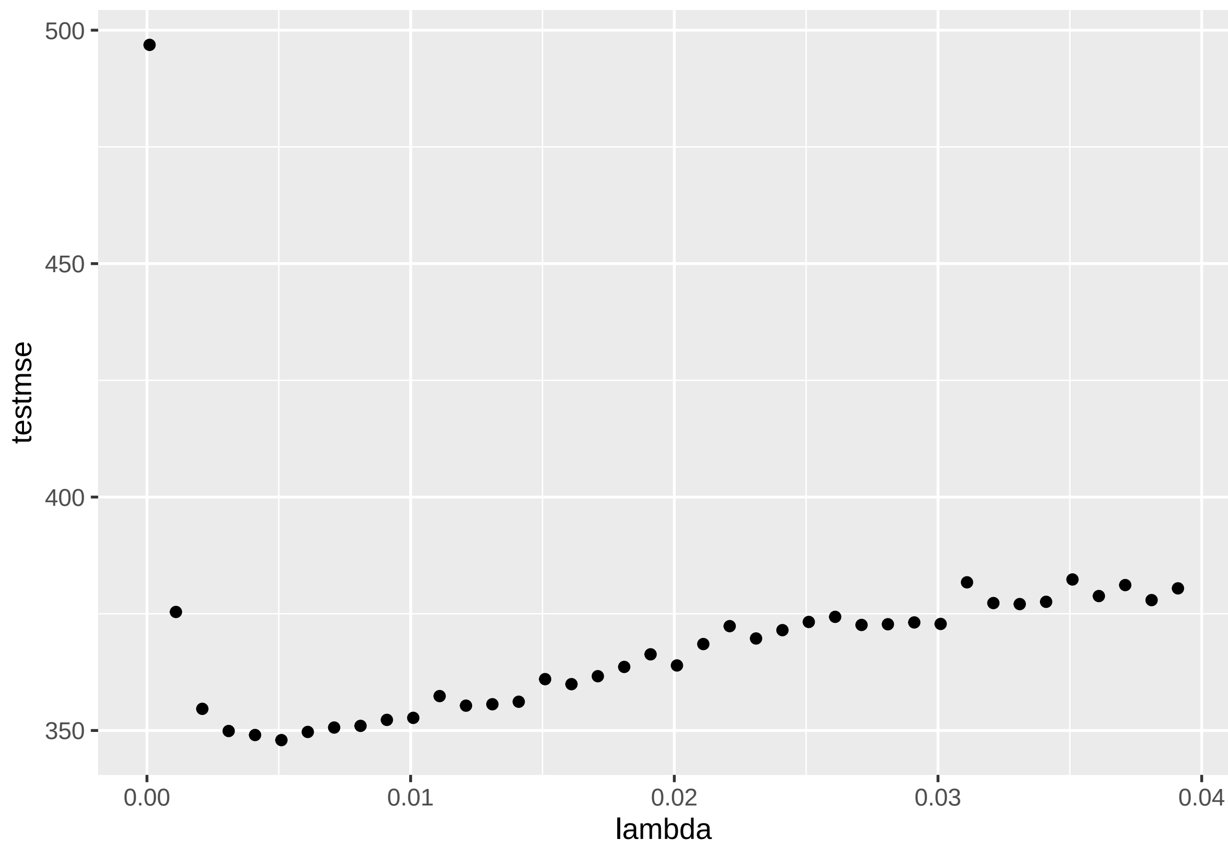
```r
train=sample(1:nrow(nesdata),1355)
full=seq(1:1807)
test=setdiff(full,train)
```

##3

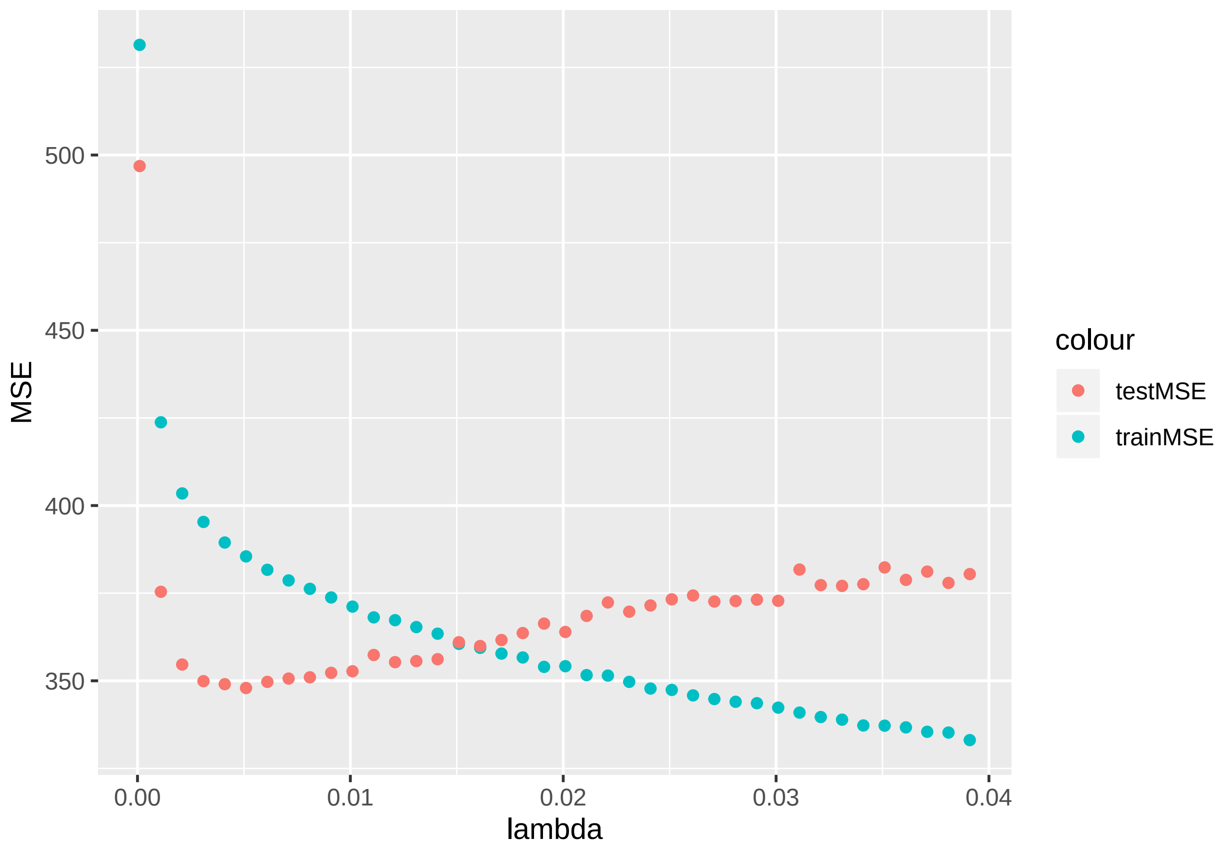
```r
library(gbm)
trainmse<-vector()
testmse<-vector()
for (i in 1:40){
boost.nesdata <- gbm(biden ~ .,
                    data=nesdata[train,],
                    distribution="gaussian",
                    n.trees=1000,
                    shrinkage=lambda[i],
                    interaction.depth = 4)
trainpred = predict(boost.nesdata, newdata = nesdata[train,],
                     n.trees = 1000)
trainmse[i]<-mean((trainpred - nesdata[train,]$biden)^2)

testpred = predict(boost.nesdata, newdata = nesdata[test,],
                    n.trees = 1000)
testmse[i] <-mean((testpred - nesdata[test,]$biden)^2)
}


ggplot()+geom_point(aes(lambda, trainmse))
```




```
ggplot()+geom_point(aes(lambda, testmse))
```




```r
msedf<-data.frame(trainmse,testmse)
ggplot(data = msedf) +
  geom_point(aes(x = lambda, y = trainmse, color = "trainMSE")) +
  geom_point(aes(x = lambda, y = testmse, color = "testMSE")) +
  ylab("MSE")
```



##4

```
trainmse2<-vector()
testmse2<-vector()
boost.nesdata2 <- gbm(biden ~ .,
                       data=nesdata[train,],
                       distribution="gaussian",
                       n.trees=1000,
                       shrinkage=0.01,
                       interaction.depth = 4)
  testpred2 = predict(boost.nesdata2, newdata = nesdata[test,],
                     n.trees = 1000)
  testmse2<-mean((testpred2 - nesdata[test,]$biden)^2)
testmse2
```

```
## [1] 353.833
```

The test MSE using $\lambda = 0.01$ is roughly equal or a little smaller than the median of test MSEs in question 3. From the plot in 3, the values of test MSEs are quite stable, which implies the test MSE values are insensitive to precise value of $\lambda = 0.01$ as long as its small enough. Because of that, the test MSE in this question falls in the stable invertal of the test MSE values shown in question 3.

##5

```
message = FALSE
warning = FALSE
```

```r
bg<-bagging(biden ~ ., nbagg = 100, data = nesdata[train,], coob=T)
testpredbg = predict(bg, newdata = nesdata[test,],
                    n.trees = 1000)
testmsebg<-  mean((testpredbg - nesdata[test,]$biden)^2)
testmsebg
```

```
## [1] 353.4455
```

##6

```r
rf <- randomForest(biden ~.,
                   data = nesdata,
                   subset = train)

testpredrf = predict(rf, newdata = nesdata[test,],
                    n.trees = 1000)
testmserf<-  mean((testpredrf - nesdata[test,]$biden)^2)
testmserf
```

```
## [1] 359.6052
```

##7

```r
model <- lm(biden ~ .,data = nesdata,subset = train)
testpredli = predict(model, newdata = nesdata[test,],
                    n.trees = 1000)
testmseli<-  mean((testpredli - nesdata[test,]$biden)^2)
testmseli
```

```
## [1] 345.1928
```

##8

Linear regression fits best because it has the smallest test MSE which means the model from the linear regression is most efficient.

Support Vector Machines

##1

```r
library(tidyverse)
library(e1071)
set.seed(2345)
trainOJ=sample(1:nrow(OJ),800)
fullOJ=seq(1:nrow(OJ))
testOJ=setdiff(fullOJ,trainOJ)
```

##2

```r
svmfit <- svm(Purchase ~ .,
              data = OJ[trainOJ,],
              kernel = "linear",
              cost = 0.01,
              scale = FALSE);
summary(svmfit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ[trainOJ, ], kernel = "linear",
##     cost = 0.01, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  618
##
##  ( 310 308 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

There are 618 support vectors.

##3

```r
train_pred <- predict(svmfit, OJ[trainOJ,])
table(predicted = train_pred, true = OJ[trainOJ,]$Purchase)
```

```
##          true
## predicted  CH  MM
##        CH 481 225
##        MM  11  83
```

```r
test_pred <- predict(svmfit, OJ[testOJ,])
table(predicted = test_pred, true = OJ[testOJ,]$Purchase)
```

```
##          true
## predicted  CH  MM
##        CH 155  71
##        MM   6  38
```

```r
test_error<-OJ[testOJ,] %>%
  # calculate estimate
  mutate(estimate = test_pred) %>%
```

```
  # calculate accuracy and convert to error
  accuracy(truth = Purchase, estimate = estimate)
#error rate
1 - test_error$.estimate[[1]]
```

```
## [1] 0.2851852
```

```
train_error<-OJ[trainOJ,] %>%
  # calculate estimate
  mutate(estimate = train_pred) %>%
  # calculate accuracy and convert to error
  accuracy(truth = Purchase, estimate = estimate)
#error rate
1 - train_error$.estimate[[1]]
```

```
## [1] 0.295
```

##4

```
tune_c <- tune(svm,
               Purchase ~ .,
               data = OJ[trainOJ,],
               kernel = "linear",
               ranges = list(cost = c(0.01,0.1,1,10,100,1000)))
summary(tune_c)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.1775
##
## - Detailed performance results:
##    cost   error dispersion
## 1 1e-02 0.18750 0.04448783
## 2 1e-01 0.18375 0.04251225
## 3 1e+00 0.17750 0.04281744
## 4 1e+01 0.18000 0.04647281
## 5 1e+02 0.18000 0.04901814
## 6 1e+03 0.17875 0.04168749
```

```
# best?
tuned_model <- tune_c$best.model
summary(tuned_model)
```

```
##
```

```
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = OJ[trainOJ,
##     ], 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:  351
##
##  ( 176 175 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

The optimal cost is 1.

##5

```
svmfit <- svm(Purchase ~ .,
              data = OJ[trainOJ,],
              kernel = "linear",
              cost = 1,
              scale = FALSE);

train_pred2 <- predict(svmfit, OJ[trainOJ,])
table(predicted = train_pred2, true = OJ[trainOJ,]$Purchase)
```

```
##          true
## predicted  CH  MM
##        CH 422  74
##        MM  70 234
```

```
test_pred2 <- predict(svmfit, OJ[testOJ,])
table(predicted = test_pred2, true = OJ[testOJ,]$Purchase)
```

```
##          true
## predicted  CH  MM
##        CH 146  19
##        MM  15  90
```

```
test_error2<-OJ[testOJ,] %>%
  # calculate estimate
  mutate(estimate = test_pred2) %>%
  # calculate accuracy and convert to error
  accuracy(truth = Purchase, estimate = estimate)
# error rate
1 - test_error2$.estimate[[1]]
```

```
## [1] 0.1259259
```

```r
train_error2<-OJ[trainOJ,] %>%
  # calculate estimate
  mutate(estimate = train_pred2) %>%
  # calculate accuracy and convert to error
  accuracy(truth = Purchase, estimate = estimate)
# error rate
1 - train_error2$.estimate[[1]]
```

```
## [1] 0.18
```

The test error rate is 12.6% and the train error rate is 18% when using the optimal cost. Optimally tuned classifer generates much smaller train and test errors than the classifer using cost=0.01 in question 3. So optimally tuned classifer has better performance.