Homework of Dataminning, CH8

Zexian Wang, Student ID 15420151152805

2017/4/24

Q9

(a)

```r
library(ISLR)
set.seed(2017)
OJ <- OJ
n <- nrow(OJ)
train = sample(n,800,replace = F)
OJ_train <- OJ[train,]
OJ_test <- OJ[-train,]
```

(b)

```r
library(tree)
oj_tree <- tree(Purchase ~ ., data = OJ_train)
summary(oj_tree)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJ_train)
## Variables actually used in tree construction:
## [1] "LoyalCH"     "PriceDiff"   "SalePriceMM"
## Number of terminal nodes:  7
## Residual mean deviance:  0.7515 = 595.9 / 793
## Misclassification error rate: 0.1525 = 122 / 800
```

The tree has 8 terminal nodes. The training error rate is 0.165

(c)

```r
oj_tree
```

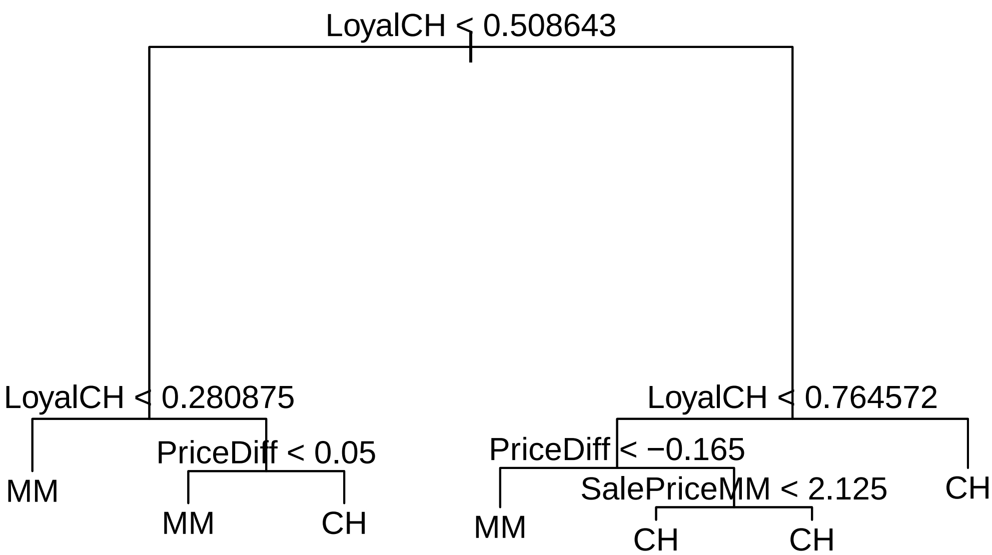
```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1071.00 CH ( 0.60875 0.39125 )
##    2) LoyalCH < 0.508643 347  403.40 MM ( 0.26801 0.73199 )
##      4) LoyalCH < 0.280875 171  119.30 MM ( 0.11111 0.88889 ) *
##      5) LoyalCH > 0.280875 176  239.50 MM ( 0.42045 0.57955 )
##       10) PriceDiff < 0.05 75   75.06 MM ( 0.20000 0.80000 ) *
##       11) PriceDiff > 0.05 101  137.10 CH ( 0.58416 0.41584 ) *
##    3) LoyalCH > 0.508643 453  350.50 CH ( 0.86976 0.13024 )
```

```
##      6) LoyalCH < 0.764572 179  204.00 CH ( 0.74302 0.25698 )
##       12) PriceDiff < -0.165 25   27.55 MM ( 0.24000 0.76000 ) *
##       13) PriceDiff > -0.165 154  143.00 CH ( 0.82468 0.17532 )
##         26) SalePriceMM < 2.125 90  102.30 CH ( 0.74444 0.25556 ) *
##         27) SalePriceMM > 2.125 64   29.93 CH ( 0.93750 0.06250 ) *
##      7) LoyalCH > 0.764572 274  104.60 CH ( 0.95255 0.04745 ) *
```

Node 8 means that, if LoyalCH < 0.0356415, the model will think the sample is belong to MM.

(d)

```r
plot(oj_tree)
text(oj_tree)
```



LoyalCH is the most important variable of the tree. If LoyalCH < 0.264232, the tree predicts MM. If LoyalCH > 0.508643, the tree predicts CH. Other cases are depended on PriceDiff and SpecialCH.

(e)

```r
oj_pred <- predict(oj_tree, OJ_test,type="class")
mt <- table(OJ_test$Purchase, oj_pred)

#table
mt
```

```
##     oj_pred
##       CH  MM
##   CH 144  22
##   MM  32  72
```

```r
#acc
(mt[1,1]+mt[2,2])/nrow(OJ_test)
```

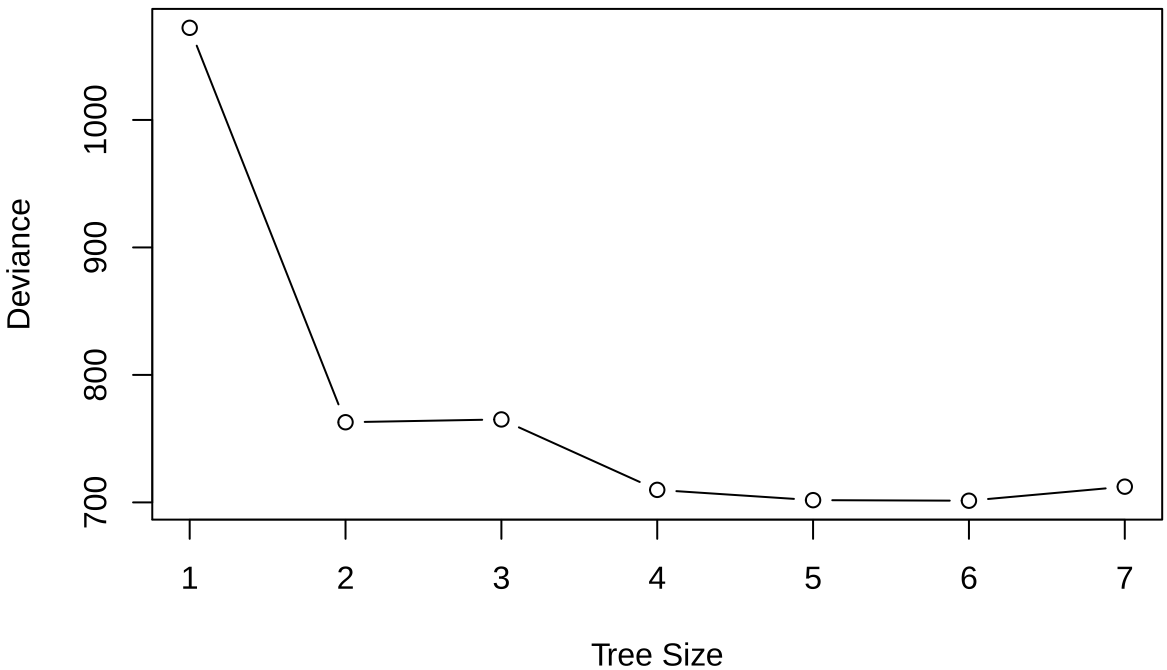
```
## [1] 0.8
```

(f)

```r
oj_cv <- cv.tree(oj_tree, FUN=prune.tree)
```

(g)

```r
plot(oj_cv$size, oj_cv$dev, type = "b", xlab = "Tree Size", ylab = "Deviance")
```



(h)

Size of 6

(i)

```r
oj_pruned <- prune.tree(oj_tree, best = 6)
```

(j)

```r
summary(oj_pruned)
```

```
##
## Classification tree:
## snip.tree(tree = oj_tree, nodes = 13L)
## Variables actually used in tree construction:
## [1] "LoyalCH"   "PriceDiff"
## Number of terminal nodes:  6
```

```
## Residual mean deviance:  0.7641 = 606.7 / 794
## Misclassification error rate: 0.1525 = 122 / 800
```

Misclassification error of pruned tree is exactly same as that of original tree.

(k)

```
pred_unpruned = predict(oj_tree, OJ_test, type = "class")
misclass_unpruned = sum(OJ_test$Purchase != pred_unpruned)
misclass_unpruned/length(pred_unpruned)
```

```
## [1] 0.2
```

```
pred_pruned = predict(oj_pruned, OJ_test, type = "class")
misclass_pruned = sum(OJ_test$Purchase != pred_pruned)
misclass_pruned/length(pred_pruned)
```

```
## [1] 0.2
```

Pruned and unpruned trees have same test error rate.

Q10

(a)

```
library(ISLR)
Hitters <- Hitters[-which(is.na(Hitters$Salary)),]
Hitters$Salary <- log(Hitters$Salary)
```

(b)

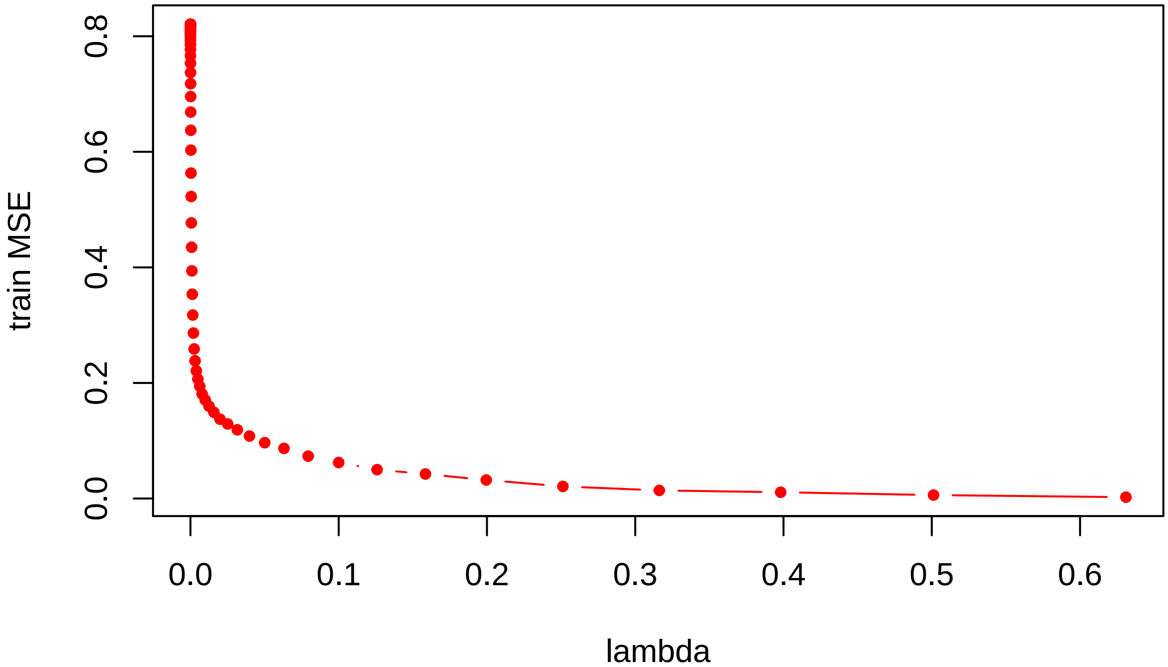
```
set.seed(2017)
n <- nrow(Hitters)
train <- sample(n,200,replace = F)
Hitters_train <- Hitters[train,]
Hitters_test <- Hitters[-train,]
```

(c)

```
set.seed(2017)
library(gbm)
```

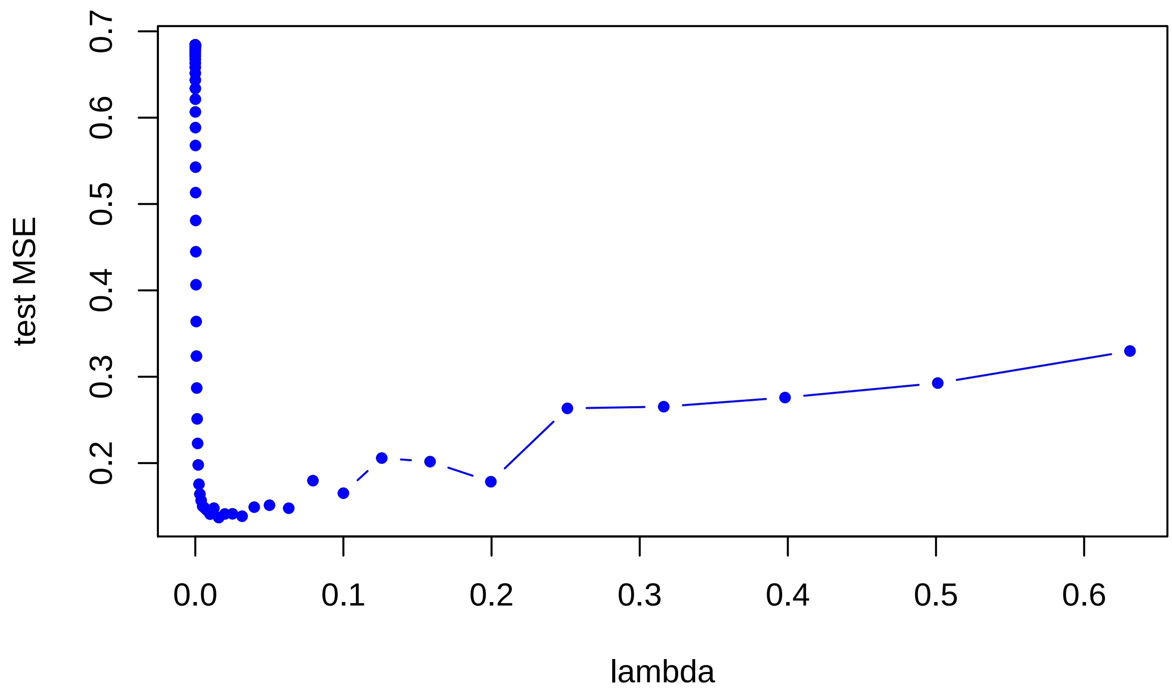
```
## Loading required package: survival

## Loading required package: lattice

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.3
```

```r
lambda <- 10^seq(-10,-0.2,by=0.1)
length_lambda <- length(lambda)
train_errors <- rep(NA,length_lambda)
test_errors <- rep(NA,length_lambda)
for (i in 1:length_lambda){
  boost <- gbm(Salary ~ ., data=Hitters_train, distribution = "gaussian", n.trees = 1000, shrinkage = la
  train_pred <- predict(boost, Hitters_train, n.trees = 1000)
  test_pred <- predict(boost, Hitters_test, n.trees = 1000)
  train_errors[i] <- mean((Hitters_train$Salary - train_pred)^2)
  test_errors[i] <- mean((Hitters_test$Salary - test_pred)^2)
}

plot(lambda, train_errors, type = "b", xlab="lambda", ylab="train MSE", col = "red", pch = 20)
```



(d)

```r
plot(lambda, test_errors, type = "b", xlab="lambda", ylab="test MSE", col = "blue", pch = 20)
```



(e)

```
lm_fit = lm(Salary ~ ., data = Hitters_train)
lm_pred = predict(lm_fit, Hitters_test)
mean((Hitters_test$Salary - lm_pred)^2)
```

```
## [1] 0.3817038
```

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-5
```

```
set.seed(2017)
x <- model.matrix(Salary ~ ., data = Hitters_train)
y <- Hitters_train$Salary
x_test <- model.matrix(Salary ~ ., data=Hitters_test)
lasso_fit <- glmnet(x,y,alpha = 1)
lasso_pred <- predict(lasso_fit, s =0.01, newx=x_test)
mean((Hitters_test$Salary - lasso_pred)^2)
```

```
## [1] 0.3529143
```

```
lambda[which.min(test_errors)]
```

```
## [1] 0.01584893
```

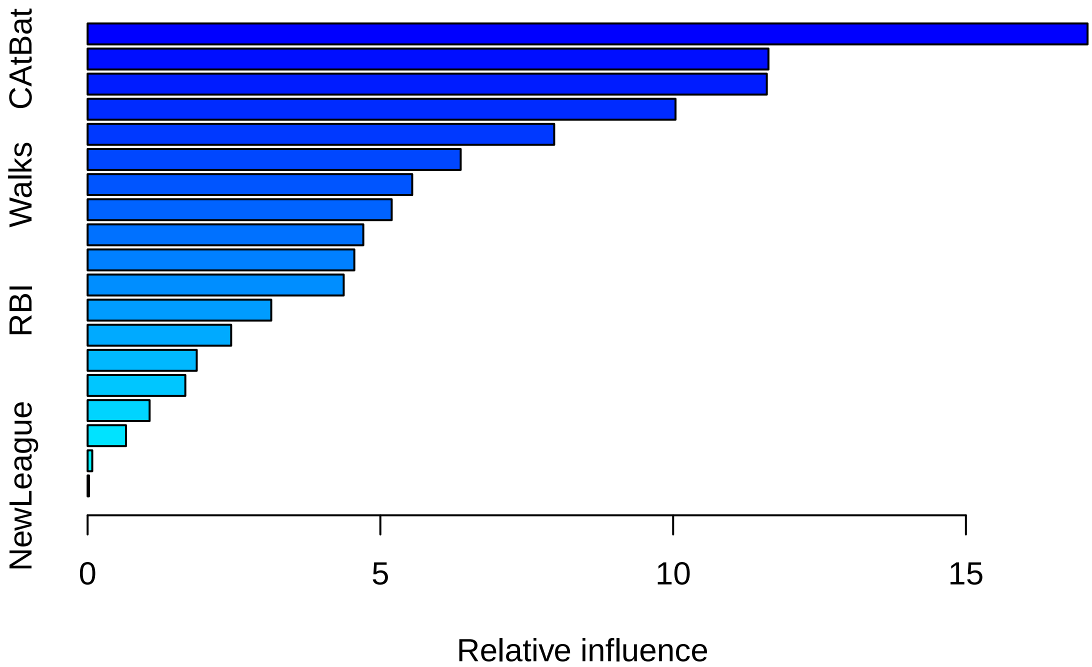
```
min(test_errors)
```

```
## [1] 0.1369177
```

Boosting has the smallest mse.

(f)

```r
boost_best = gbm(Salary ~ ., data = Hitters_train, distribution = "gaussian",
    n.trees = 1000, shrinkage = lambda[which.min(test_errors)])
summary(boost_best)
```




```
##                 var     rel.inf
## CRuns         CRuns 17.07760363
## CAtBat       CAtBat 11.62796113
## CHits         CHits 11.59943525
## CRBI           CRBI 10.04073696
## CWalks       CWalks  7.96891965
## Years         Years  6.37175371
## Walks         Walks  5.54389378
## PutOuts     PutOuts  5.19348620
## Hits           Hits  4.70871881
## HmRun         HmRun  4.55541210
## CHmRun       CHmRun  4.37351601
## RBI             RBI  3.13664348
## Errors       Errors  2.45198921
## AtBat         AtBat  1.86374895
## Runs           Runs  1.66928038
## Assists     Assists  1.05904168
## Division   Division  0.65517031
## League       League  0.07965166
## NewLeague NewLeague  0.02303710
```

CRuns, CAtBat and CHits.

(g)

```r
library(randomForest)
```

```
## randomForest 4.6-12
```

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

```r
set.seed(2017)
rf <- randomForest(Salary ~ ., data = Hitters_train, ntree = 500, mtry=19)
rf_pred <- predict(rf,Hitters_test)
mean((Hitters_test$Salary - rf_pred)^2)
```

```
## [1] 0.1127392
```