Math 7553 – Spring 2018

HW #3 (Hand In)

Shuddha Chowdhury

Submission Date: 04/03/2018

Chapter 8: Exercise 8

In the lab, a classification tree was applied to the Carseats data set after converting Sales into a qualitative response variable. Now we will seek to predict Sales using regression trees and related approaches, treating the response as a quantitative variable.

Split the data set into a training set and a test set.

Ans:

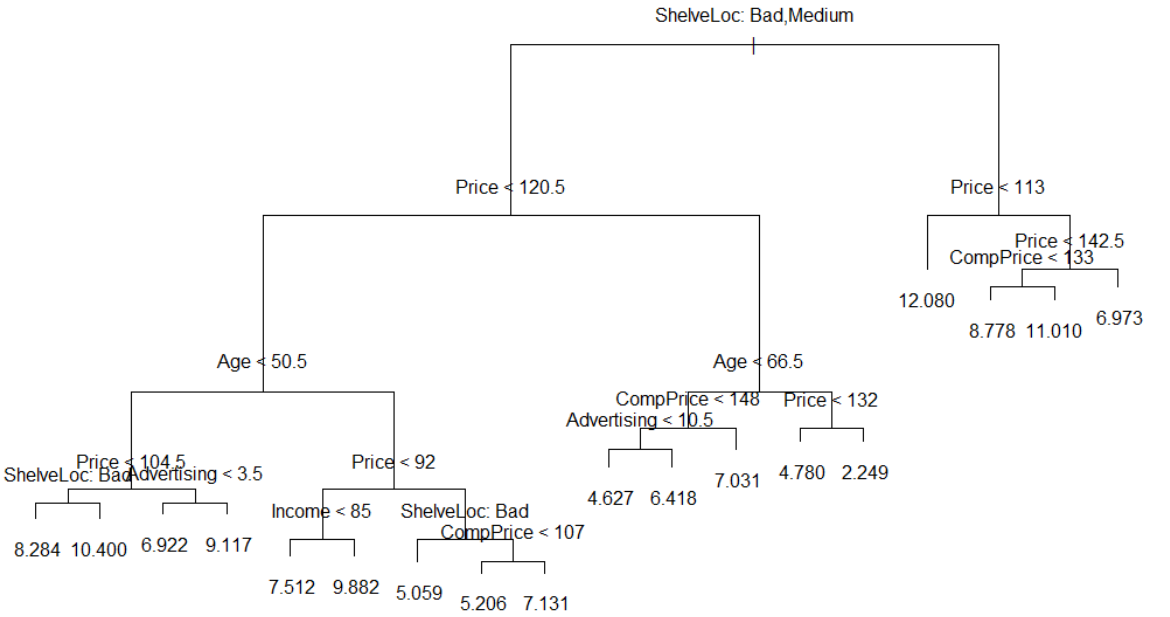
```
library(ISLR)
attach(Carseats)
set.seed(1)


#splitting the data into training and test data set


train = sample(dim(Carseats)[1], dim(Carseats)[1]/2)
Carseats.train = Carseats[train, ]
Carseats.test = Carseats[-train, ]
```

(b) Fit a regression tree to the training set. Plot the tree, and interpret the results. What test error rate do you obtain?

Ans :

```
library(tree)
tree.carseats = tree(Sales ~ ., data = Carseats.train)
summary(tree.carseats)

##
## Regression tree:
## tree(formula = Sales ~ ., data = Carseats.train)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Age"         "Advertising" "Income
"
## [6] "CompPrice"
## Number of terminal nodes:  18
## Residual mean deviance:  2.36 = 429.5 / 182
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -4.2570 -1.0360  0.1024  0.0000  0.9301  3.9130
```

```
plot(tree.carseats)
text(tree.carseats, pretty = 0)
```



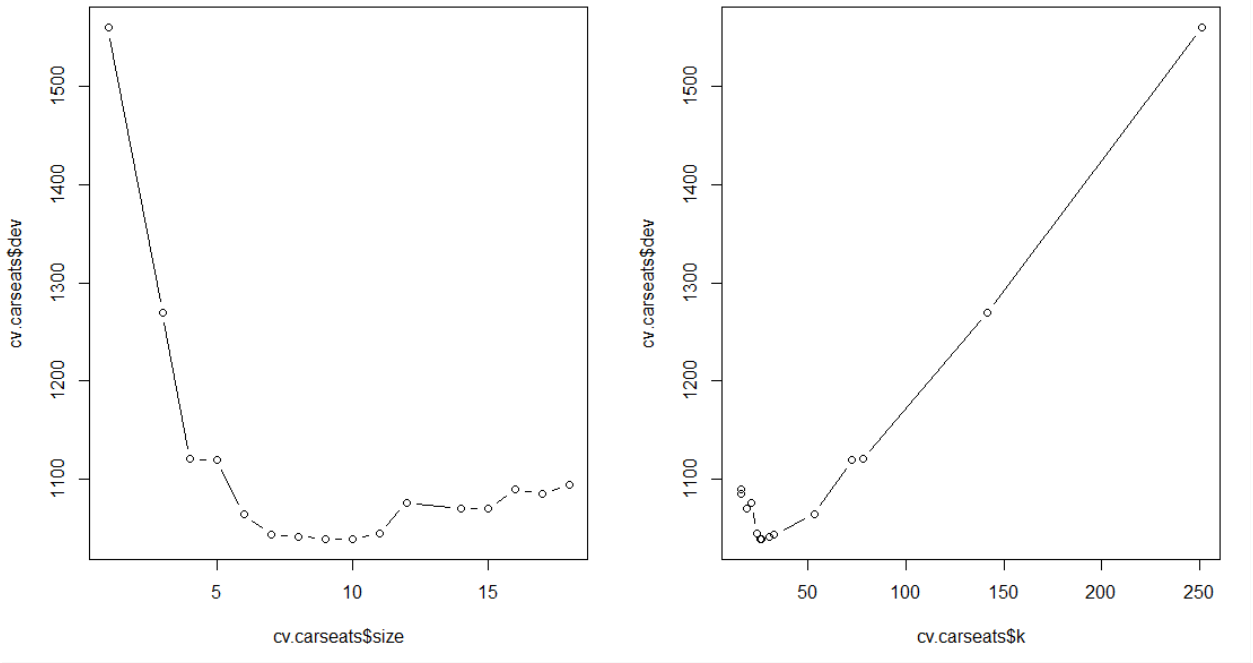

```
pred.carseats = predict(tree.carseats, Carseats.test)
mean((Carseats.test$Sales - pred.carseats)^2)

## [1] 4.148897
```

```
# Reference - https://cran.r-project.org/web/packages/tree/tree.pdf
      https://www.rdocumentation.org/packages/tree/versions/1.0-37
```

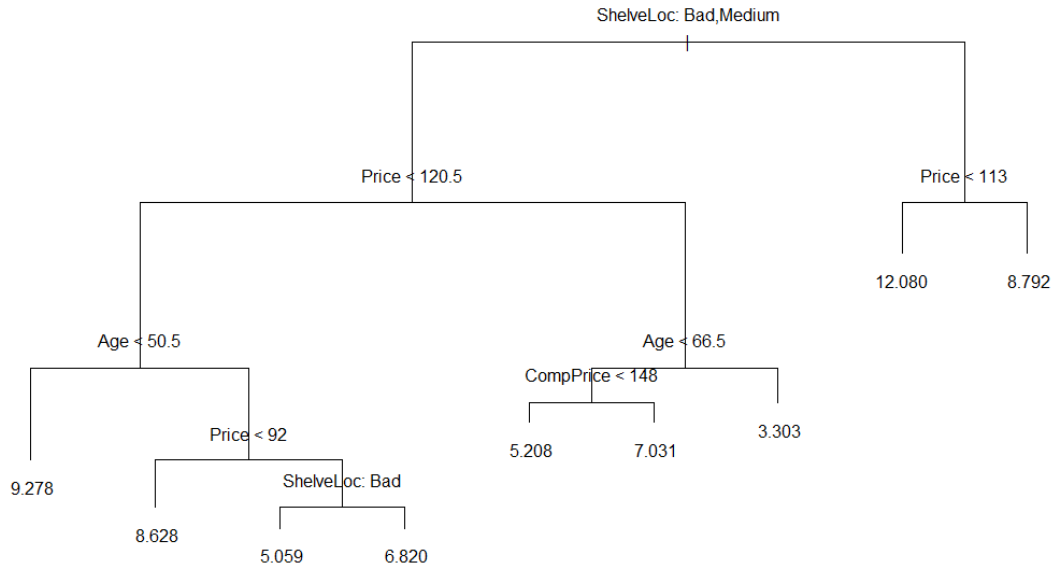
(c) Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test error rate?

Ans:

```
cv.carseats = cv.tree(tree.carseats, FUN = prune.tree)
par(mfrow = c(1, 2))
plot(cv.carseats$size, cv.carseats$dev, type = "b")
plot(cv.carseats$k, cv.carseats$dev, type = "b")
```




```
pruned.carseats = prune.tree(tree.carseats, best = 9)
par(mfrow = c(1, 1))
plot(pruned.carseats)
text(pruned.carseats, pretty = 0)
```




```
pred.pruned = predict(pruned.carseats, Carseats.test)
mean((Carseats.test$Sales - pred.pruned)^2)
```

```
## [1] 4.993124
```

When we prune the tree, we noticed that the test MSE increases to 4.99.

(d) Use the bagging approach in order to analyze this data. What test error rate do you obtain? Use the importance() function to determine which variables are most important.

Ans :

```
library(randomForest)

## Warning: package 'randomForest' was built under R version 3.4.4

## randomForest 4.6-14

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

bag.carseats = randomForest(Sales ~ ., data = Carseats.train, mtry = 1
0, ntree = 500, importance = T)
bag.pred = predict(bag.carseats, Carseats.test)
mean((Carseats.test$Sales - bag.pred)^2)

## [1] 2.633915

importance(bag.carseats)

##                %IncMSE IncNodePurity
## CompPrice   16.9874366    126.852848
## Income       3.8985402     78.314126
## Advertising 16.5698586    123.702901
## Population   0.6487058     62.328851
## Price       55.3976775    514.654890
## ShelveLoc   42.7849818    319.133777
## Age         20.5135255    185.582077
## Education    3.4615211     42.253410
## Urban       -2.5125087      8.700009
## US           7.3586645     18.180651
```

Here after performing bagging, we see that the improved test MSE 2.63. Here we notice Price, ShelveLoc and Age are three most important predictor variables of Sale.

Reference: https://www.r-bloggers.com/random-forests-in-r/

https://cran.r-project.org/web/packages/randomForest/randomForest.pdf

(e) Use random forests to analyze this data. What test error rate do you obtain? Use the importance () function to determine which variables are most important. Describe the effect of m, the number of variables considered at each split, on the error rate obtained.

Ans :

```
rf.carseats = randomForest(Sales ~ ., data = Carseats.train, mtry = 5,
ntree = 500, importance = T)
rf.pred = predict(rf.carseats, Carseats.test)
mean((Carseats.test$Sales - rf.pred)^2)

## [1] 2.816693
```

```
importance(rf.carseats)

##                %IncMSE IncNodePurity
## CompPrice   11.304167     126.68519
## Income       5.423214     102.44073
## Advertising 13.351166     137.98835
## Population   1.131119      82.24483
## Price       46.600559     451.70021
## ShelveLoc   37.352447     278.79756
## Age         19.992113     194.99430
## Education    1.945616      51.70741
## Urban       -2.244558      10.87383
## US           6.261365      20.83998
```

Here we notice that random forest lowers the MSE on the test set data to 2.81. If we change m then test MSE varies from 2.6 to 3. Again we notice that ShelveLoc and Age are three most important predictors of Sale.

Chapter 8: Exercise 10

We now use boosting to predict Salary in the Hitters data set.

(a) Remove the observations for whom the salary information is unknown, and then log-transform the salaries.

Ans:

```
library(ISLR)
sum(is.na(Hitters$Salary))
```

```
## [1] 59
```

```
Hitters = Hitters[-which(is.na(Hitters$Salary)), ]
sum(is.na(Hitters$Salary))
```

```
## [1] 0
```

```
Hitters$Salary = log(Hitters$Salary)
```

(b) Create a training set consisting of the first 200 observations, and a test set consisting of the remaining observations.

Ans:

```
train = 1:200
```

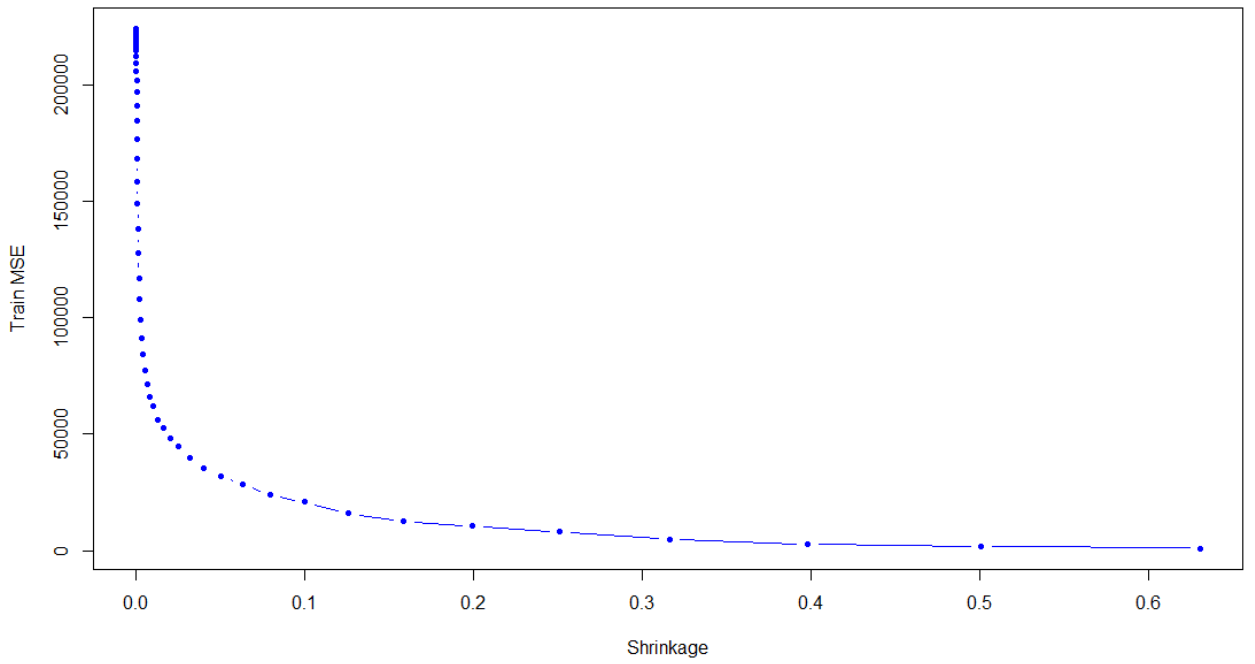
```
Hitters.train = Hitters[train, ]
```

```
Hitters.test = Hitters[-train, ]
```

c) Remove the observations for whom the salary information is unknown, and then log-transform the salaries.

Ans:

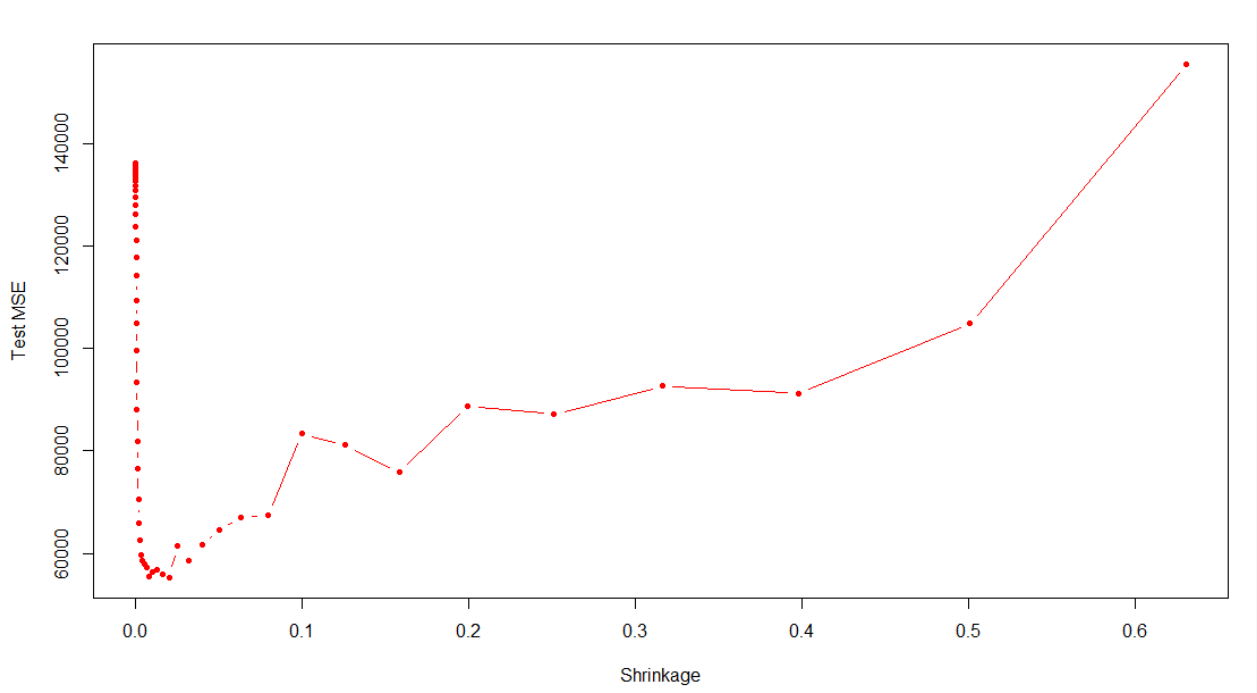
```
set.seed(103)
pows = seq(-10, -0.2, by = 0.1)
lambdas = 10^pows
length.lambdas = length(lambdas)
train.errors = rep(NA, length.lambdas)
test.errors = rep(NA, length.lambdas)
for (i in 1:length.lambdas) {
  boost.hitters = gbm(Salary ~ ., data = Hitters.train, distribution =
"gaussian",  n.trees = 1000, shrinkage = lambdas[i])
  train.pred = predict(boost.hitters, Hitters.train, n.trees = 1000)
  test.pred = predict(boost.hitters, 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(lambdas, train.errors, type = "b", xlab = "Shrinkage", ylab = "Tr
ain MSE",
     col = "blue", pch = 20)
```



(d) Produce a plot with different shrinkage values on the x-axis and the corresponding test set MSE on the y-axis.

Ans:

```
plot(lambdas, test.errors, type = "b", xlab = "Shrinkage", ylab = "Tes
t MSE", col = "red", pch = 20)
```




```
min(test.errors)
```

```
## [1] 0.2560507
```

```
lambdas[which.min(test.errors)]
```

```
## [1] 0.05011872
```

Here the Minimum test error is obtained at λ=0.05.

e) Compare the test MSE of boosting to the test MSE that results from applying two of the regression approaches seen in Chapters 3.

Ans:

```
lm.fit = lm(Salary ~ ., data = Hitters.train)
lm.pred = predict(lm.fit, Hitters.test)
mean((Hitters.test$Salary - lm.pred)^2)
```

```
## [1] 0.4917959
```

```
set.seed(134)
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] 0.4701
```

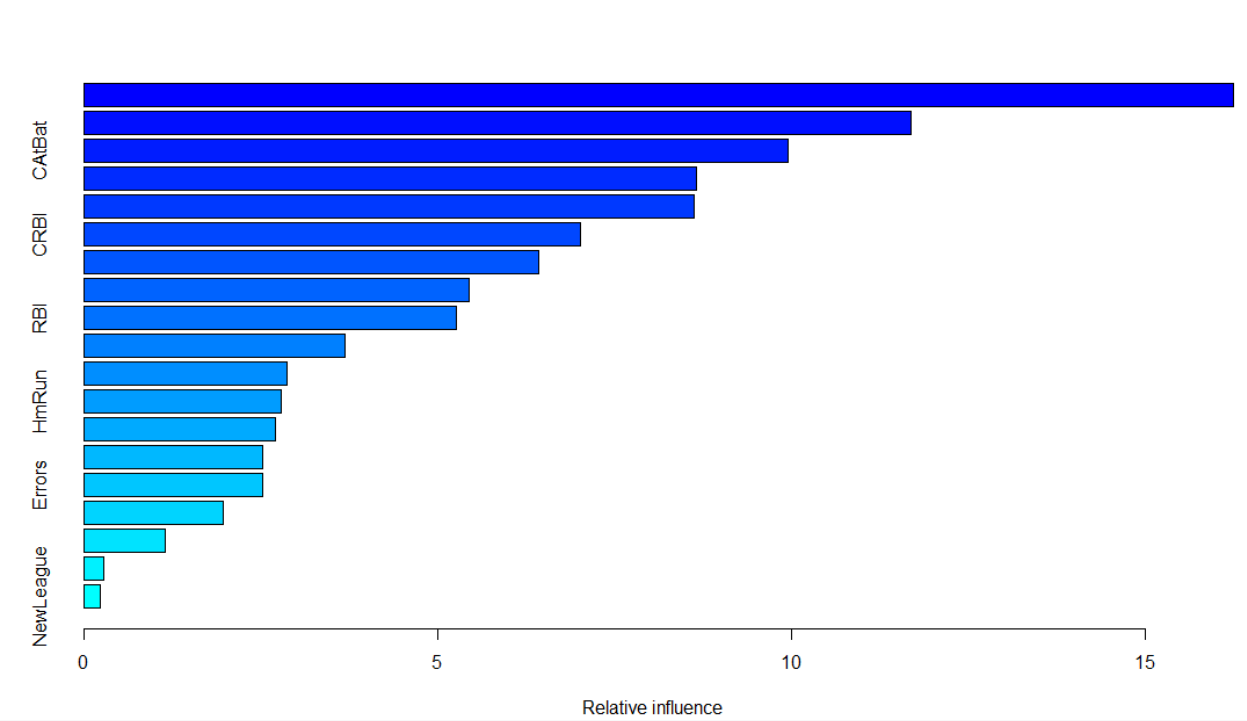
Here both linear model and regularization like Lasso have higher test MSE value than boosting value.

(f) Which variables appear to be the most important predictors in the boosted model?

Ans:

```
boost.best = gbm(Salary ~ ., data = Hitters.train, distribution = "gaussian",
n.trees = 1000, shrinkage = lambdas[which.min(test.errors)])
```

```
summary(boost.best)
```




```
  var       rel.inf
CHmRun       CHmRun 16.2464377
Walks         Walks 11.6816442
CAtBat       CAtBat  9.9537777
Years         Years  8.6510975
Hits           Hits  8.6259821
CRBI           CRBI  7.0184910
PutOuts     PutOuts  6.4172370
CWalks       CWalks  5.4451756
RBI             RBI  5.2644794
CHits         CHits  3.6809891
```

```
AtBat         AtBat  2.8699489
HmRun         HmRun  2.7874174
CRuns         CRuns  2.7066622
Assists     Assists  2.5190996
Errors       Errors  2.5151030
Division   Division  1.9650221
Runs           Runs  1.1423270
League       League  0.2773784
NewLeague NewLeague  0.2317301
```

We notice that CAtBat, CRBI and CWalks are three most important variables in that order.