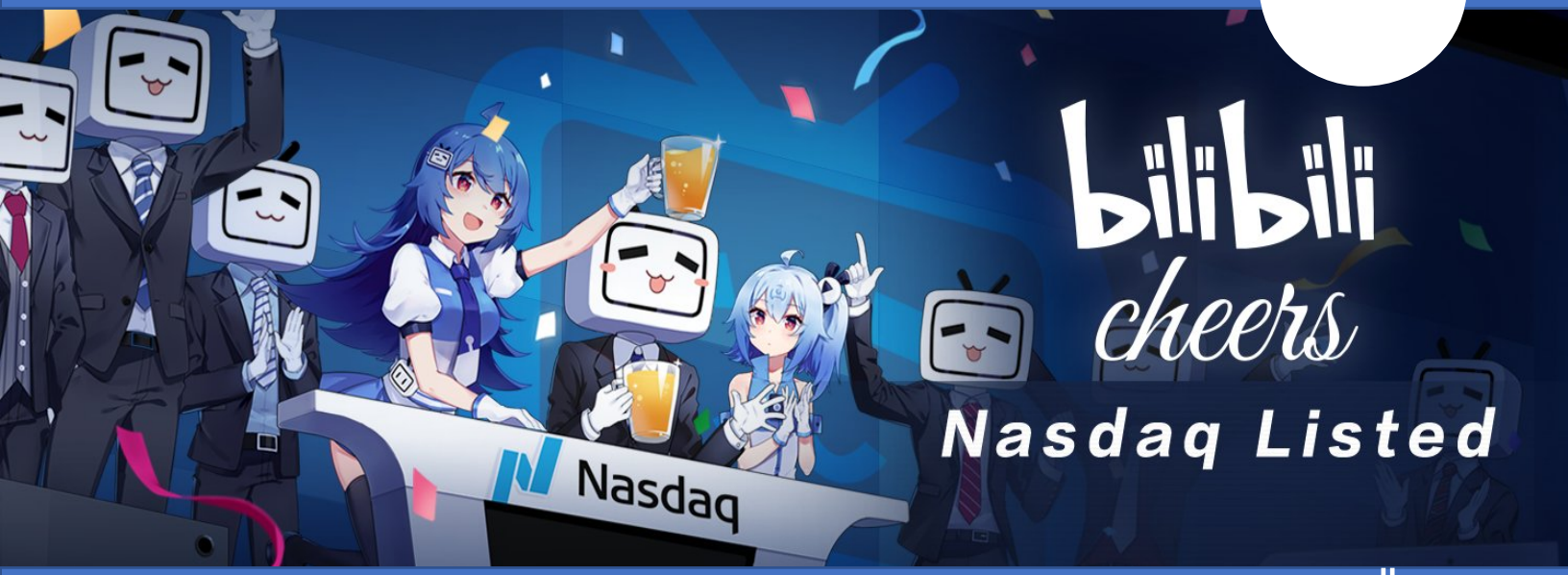


Analyze Nowadays, Predict the Future

Xianglin Chen

Baruch College

CIS3920-S2CA

Summer 2021

Picture Source: “Chinese Audio and Video Creation Platform Bilibili Experiences Temporary Inaccessibility Due to Engine Room Failure”. *Pandaily*. July 14 2021 . https://pandaily.com/chinese-audio-and-video-creation-platform-bilibili-experiences-temporary-inaccessibility-due-to-engine-room-failure/.

Contents

Summary

It is very hard for people in China mainland to watch YouTube due to governmental policies. At this time, the company named Bilibili Inc, a video site in China, opens to users who want to upload and view the video in various areas such as animation, game, or vlog. Its stock symbol is BILI. It used to be a community focused on young age who love animation and dramas. As the company has developed, the age range has become wider, and more people joined to create high-quality video products.

In this project, I am going to self-study about the classification and regression trees in Chapter 8. Then, I will apply the methods I learned to the BILI risk data set by using the "Train" to predict the "Test" set. I will not only calculate the accuracy rate, but also create a classification space plot using the methodology I chose.

In the end, to find the best method, I will use the Cardiac data set to compare the different of the performance of KNN, Naïve Bayes, Logistic Regression (with bestglm) and my method (Boosting Regression) from what I have learned.

The appendix is at the end of the project. It shows I have processed all the notes on the ISLR Textbook. The following is the dictionary of my Workplace history.

```
> ls()
 [1] "Age"           "b"             "bag.boston"    "best.out"      "bili.test"     "biliCART"
 [7] "bilirisk"      "BILIrisk"      "BILIriskNew"   "BILIsecondtest" "BILIthirdtest" "bilitrain"
[13] "BILIy"         "boost.bili"    "boost.boston"  "boost.cardiac" "boston.test"   "Byhat"
[19] "Cardiac"       "Cardiac1"      "Carseats"      "Carseats.test" "cc"            "Cforecast"
[25] "Cglm"          "cl"            "Cmodel"        "Cprobs"        "Ctest"         "CTest.X"
[31] "CTest.Y"       "CTrain"        "CTrain.X"      "CTrain.Y"      "cv.boston"     "cv.carseats"
[37] "Cyhat"         "Cyhat1"        "High"          "High.test"     "Holdk"         "k"
[43] "kcvSearch"     "knn.pred"      "kSearch"       "L"             "M"             "mbp"
[49] "model"         "N"             "Nbilitrain"    "NewBILIrisk"   "Newbilitrain"  "Predict"
[55] "ProbeGlm"      "prune.carseats" "rf.boston"    "Risk"          "s"             "sim.Values"
[61] "Sim.Values"    "StBILI"        "StPrX"         "table"         "Test"          "train"
[67] "Train"         "Train.Y"       "tree.boston"   "tree.carseats" "tree.pred"     "TRisk"
[73] "w"             "x"             "y"             "yhat"          "yhat.bag"      "yhat.bili"
[79] "yhat.boost"    "yhat.rf"
```

Part A

A1.

I will walk through the stock risk data at the level of detail used in ISLR. It means I will get through the boosting part (8.3.4 in appendix) by using the BILI risk data. The BILI risk data set contains total 10 lags and 1 column for determination if it is High Risk or Low risk for the daily returns. I also transformed the first column "Risk" to be data frame instead of character.

```
> setwd("/Users/xianglinchen/Desktop/CIS3920/Week3/HM")
> BILIrisk=read.csv("BILIlag10.csv")
> head(BILIrisk)
      Risk Lag1 Lag2 Lag3 Lag4 Lag5  Lag6  Lag7  Lag8
1 Low Risk 0.41 0.56 0.34 0.73 0.18 0.299 0.530 0.709
2 Low Risk 0.37 0.41 0.56 0.34 0.73 0.180 0.299 0.530
3 Low Risk 0.17 0.37 0.41 0.56 0.34 0.730 0.180 0.299
4 Low Risk 0.24 0.17 0.37 0.41 0.56 0.340 0.730 0.180
5 Low Risk 0.21 0.24 0.17 0.37 0.41 0.560 0.340 0.730
6 Low Risk 0.21 0.21 0.24 0.17 0.37 0.410 0.560 0.340
   Lag9 Lag10
1 0.590 1.150
2 0.709 0.590
3 0.530 0.709
4 0.299 0.530
5 0.180 0.299
6 0.730 0.180
```

```
> dim(BILIrisk)
[1] 820  11
> TRisk=rep(1,820)
> TRisk[BILIrisk$Risk=="Low Risk"]=0
> BILIriskNew=data.frame(BILIrisk[,-1],TRisk)
> head(BILIriskNew)
  Lag1 Lag2 Lag3 Lag4 Lag5  Lag6  Lag7  Lag8  Lag9 Lag10
1 0.41 0.56 0.34 0.73 0.18 0.299 0.530 0.709 0.590 1.150
2 0.37 0.41 0.56 0.34 0.73 0.180 0.299 0.530 0.709 0.590
3 0.17 0.37 0.41 0.56 0.34 0.730 0.180 0.299 0.530 0.709
4 0.24 0.17 0.37 0.41 0.56 0.340 0.730 0.180 0.299 0.530
5 0.21 0.24 0.17 0.37 0.41 0.560 0.340 0.730 0.180 0.299
6 0.21 0.21 0.24 0.17 0.37 0.410 0.560 0.340 0.730 0.180
  TRisk
1     0
2     0
3     0
4     0
5     0
6     0
```

In the boosting part, I am going to use the gbm package which is the gbm() function to fit boosted regression trees to the BILI data set. However, before I went through the gbm() part, I need to set up what will be the "train". The train will use sample to randomly choose column from the data set.

```
> bilitrain=sample(1:nrow(BILIrisk),nrow(BILIrisk)/2)
```

Since there's regression problem, I will use "distribution= "gaussian" in the argument. The argument n.trees=8000 indicates that I want 8000 trees and the option intereaction.depth=4 limits the depth of the tree.

```
> library(gbm)
> set.seed(1)
> boost.bili=gbm(TRisk~.,data=BILIriskNew[bilitrain,],distribution="gaussian",n.tree=8000,interaction.depth=5)
> summary(boost.bili)
        var     rel.inf
Lag1   Lag1 98.69494127
Lag2   Lag2  0.25753342
Lag10 Lag10  0.23533419
Lag5   Lag5  0.19640166
Lag6   Lag6  0.13682107
Lag3   Lag3  0.12582221
Lag4   Lag4  0.10047589
Lag8   Lag8  0.09337130
Lag7   Lag7  0.08904340
Lag9   Lag9  0.07025559
```

I can know that Lag1 is the most related variable, which has rel.inf of approximately 98.69, is much more than the other variables. The following will be Lag 2 with 0.25.

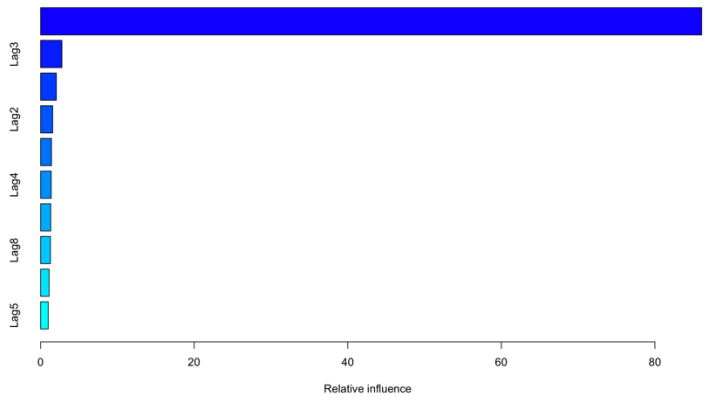
I found that it is too obvious when the data for risk is 0 when “Low Risk” and 1 when “High Risk”. So, I tried to make Risk column as the difference between the median and daily returns. I believe that it will be more detail oriented for the risk levels. Therefore, I reset the BILIrisk data set. Of course, when the Risk value is higher than 0, it means the range is higher than the median, and it will be “High Risk”; in the opposite side, when the Risk value is lower than 0, the range is lower than the median, and it will be “Low Risk”.

Risk	Lag1	Lag2	Lag3	Lag4	Lag5	Lag6	Lag7	Lag8	Lag9	Lag10
-0.729999	0.41	0.56	0.34	0.73	0.18	0.299	0.53	0.709	0.59	1.15
-0.769999	0.37	0.41	0.56	0.34	0.73	0.18	0.299	0.53	0.709	0.59
-0.969999	0.17	0.37	0.41	0.56	0.34	0.73	0.18	0.299	0.53	0.709

The summary went different when the Risk variable change to the specific number instead of 1 or 0. But the highest related variable is still Lag1 and the second is Lag3. In my opinion, the second data set will be more accurate.

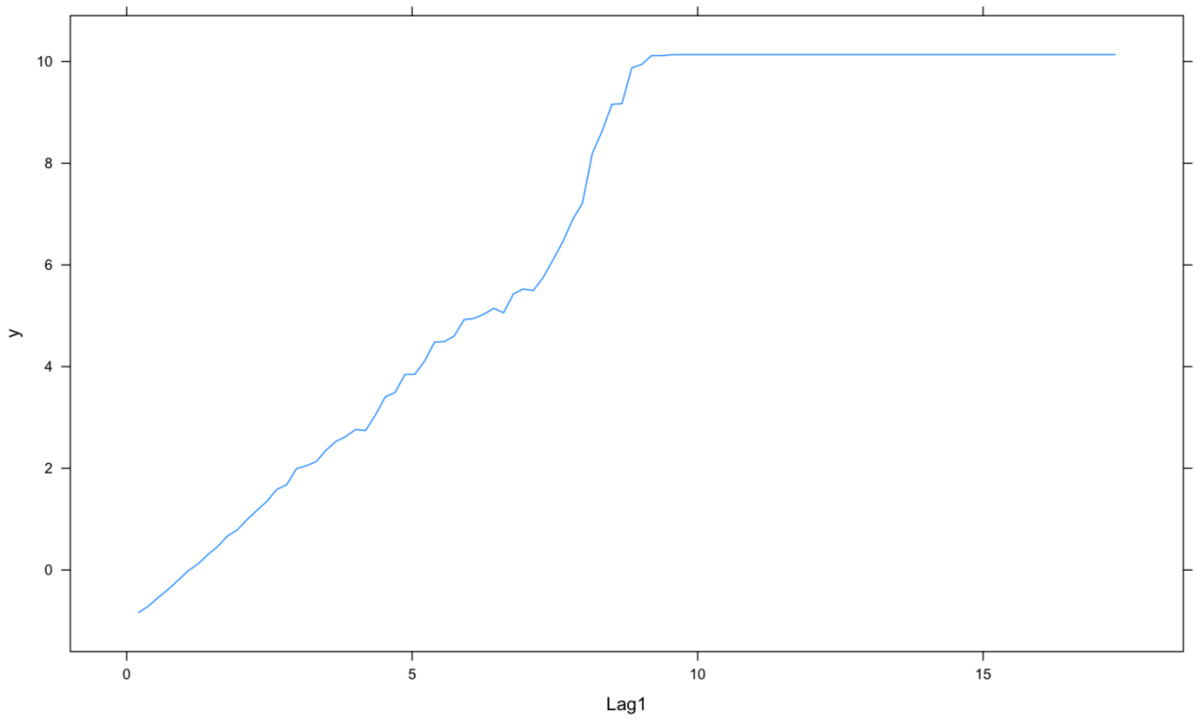
```
> NewBILIrisk=read.csv("NewBILIlag10.csv")
> Newbilitrain=sample(1:nrow(NewBILIrisk),nrow(NewBILIrisk)/2)
>
boost.bili=gbm(Risk~.,data=NewBILIrisk[Newbilitrain,],distribution="ga
ussian",n.tree=8000,interaction.depth=5)
```

```
> summary(boost.bili)
        var    rel.inf
Lag1   Lag1 86.0893528
Lag3   Lag3  2.7765503
Lag6   Lag6  2.0525648
Lag2   Lag2  1.5723692
Lag10 Lag10  1.4147815
Lag4   Lag4  1.3805787
Lag7   Lag7  1.3234446
Lag8   Lag8  1.2799275
Lag9   Lag9  1.1119716
Lag5   Lag5  0.9984591
```



I tried to plot the graph to produce partial dependence plots for the most related variables. In this case, I found that the risk increases with Lag1.

```
> par(mfrow=c(1,2))
> plot(boost.bili,i="Lag1")
```



Then, I use the boosted model to predict Risk on the test value. The test MSE obtain 0.9299 which is very low for the model.

```
> yhat.bili=predict(boost.bili,newdata=NewBILIrisk[-
Newbilitrain,],n.tree=8000)
> bili.test=NewBILIrisk[-Newbilitrain,"Risk"]

> mean((yhat.bili-bili.test)^2)
[1] 0.9298815
```

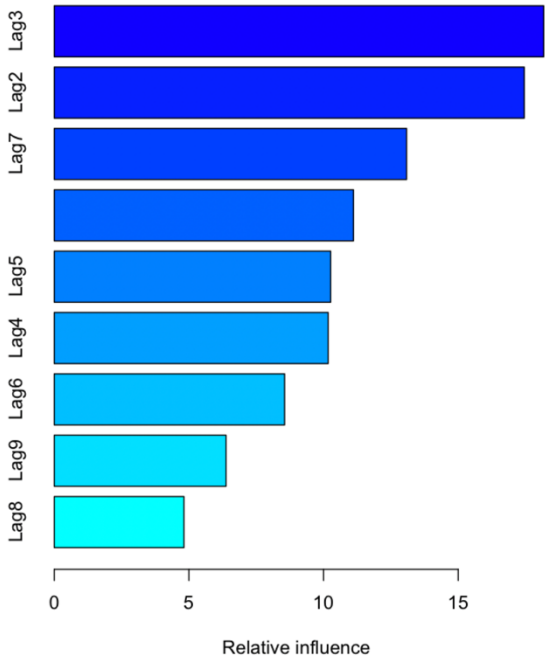
Then I added the shrinkage to the boost.bili. The result shows that the new/higher shrinkage did decrease the MSE for the model.

```
>
boost.bili=gbm(Risk~.,data=NewBILIrisk[Newbilitrain,],distribution="gaussian",n.
tree=8000,interaction.depth=5,shrinkage=0.2,verbose=F)
> yhat.bili=predict(boost.bili,newdata=NewBILIrisk[-Newbilitrain,],n.tree=8000)
> mean((yhat.bili-bili.test)^2)
[1] 0.8702865
```

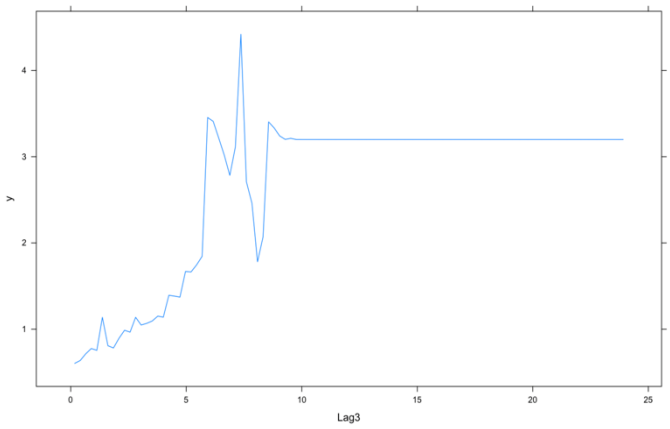
But I was still not satisfied with the result. I think Lag1 is too obvious that has a strong relationship with the risk value. I want to know more about the relationship between other lags and the risk value. So, I redo the test without Lag1. The result was surprising.

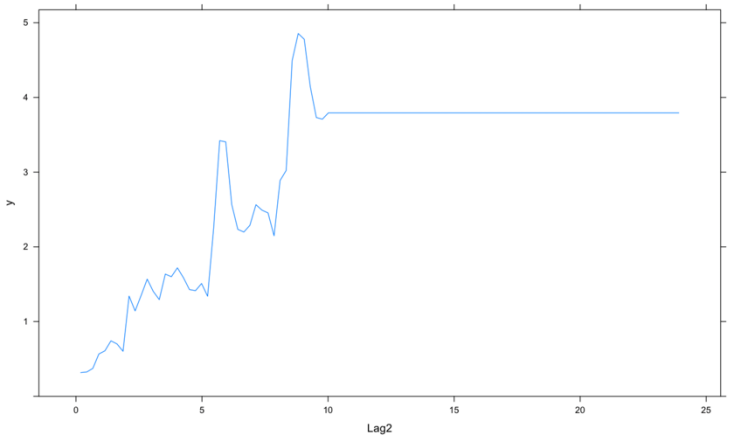
This time, the most related variables are first Lag3 and then Lag2. The rel.inf for these two variables are not same as the Lag1 in the previous prediction. Both has a value around 20.

```
> bilirisk=data.frame(NewBILIrisk[3:11],NewBILIrisk$Risk)
> Nbilitrain=sample(1:nrow(bilirisk),nrow(bilirisk)/2)
>
boost.bili=gbm(NewBILIrisk.Risk~.,data=bilirisk[Nbilitrain,],distribution="gauss
ian",n.tree=8000,interaction.depth=5)
> summary(boost.bili)
        var   rel.inf
Lag3   Lag3 18.173109
Lag2   Lag2 17.451424
Lag7   Lag7 13.083162
Lag10 Lag10 11.110201
Lag5   Lag5 10.264055
Lag4   Lag4 10.170787
Lag6   Lag6  8.552868
Lag9   Lag9  6.378672
Lag8   Lag8  4.815722
```



I also did the graph for Lag 3 and Lag 2. Like the Lag1 graph, the risk increases with Lag3 and Lag2.





I calculated the MSE for this time and I got 3.5601. Although the MSE is higher, but I think I found a more supervising result about Lag3 and Lag2, even Lag7.

```
> yhat.bili=predict(boost.bili,newdata=bilirisk[-Nbilitrain,],n.tree=8000)
> mean((yhat.bili-bili.test)^2)
[1] 3.560118
```

Then, I changed to different shrinkage parameter and added the verbose to decrease the MSE. Finally, I get 3.4566 which prove that the shrinkage did slightly lower test MSE than the default shrinkage.

```
>
boost.bili=gbm(NewBILIrisk.Risk~.,data=bilirisk[Nbilitrain,],distribution="gauss
ian",n.tree=8000,interaction.depth=5,shrinkage=0.2,verbose=F)
> yhat.bili=predict(boost.bili,newdata=bilirisk[-Nbilitrain,],n.tree=8000)
> mean((yhat.bili-bili.test)^2)
[1] 3.456616
```

A2.

Now, I turn to the challenging methodology in chapter 8(page312). This page talks about the *Gini index* which is defined by

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

(ISLR Text page 312)

It is a challenging part since there's not much explanation about it. Instead, the alternative of the Gini index, cross-entropy expends in the Textbook. Therefore, I did some research to understand what its usage is.

On the internet, the *Gini index* is generally defined as a measure of income distribution across a population. The larger *Gini index* means the higher inequality on the distribution of the income.

Generally, the *Gini index* measures the probability, which is the classification error rate, of a variable be wrongly classified under a randomly choose. The ISLR text explains that the *Gini index* is "a measure of total variance across the L classes".

To use *the Gini index*, node purity is important. Or we can say that the *Gini* index is based on the *Gini purity.* The degree of the impurity index is between 0 and 1. When 0 means elements belong to one category, 1 means element belong to various categories. On the above equation, Pmk means the probability of category K having class m, and (1-Pmk) probability of category K not having class m. Therefore, it means when the larger the index is, the closer between (1-Pmk) and Pmk, and the more "inequality" as mentioned above is.

Then, I try to practice the *Gini index* to better understand its function. From the various research, I learned that the "ineq" package is used for the *Gini index.* So, I download it to the R.

```
> install.packages("ineq")
trying URL 'https://cran.irsn.fr/bin/macosx/contrib/4.1/ineq_0.2-13.tgz'
Content type 'application/x-gzip' length 86567 bytes (84 KB)
==================================================
downloaded 84 KB


The downloaded binary packages are in
   /var/folders/21/6jktflrn6_33bfrvbvjx3x7r0000gn/T//RtmpVLDvBT/
downloaded_packages
> library(ineq)
```

I tried to calculate the Gini for the BILIriskNew data set which place the "High Risk" as 1 and "Low Risk" as 0. The result for the Trisk column was 0.4976 and for Lag1 is 0.5438.

```
> ineq(BILIriskNew$TRisk,type="Gini")
[1] 0.497561
> ineq(BILIriskNew$Lag1,type="Gini")
[1] 0.5437729
```

I searched that the CART (Classification Regression Tree) is using the *Gini Index* which is more efficient. From a YouTube video "4.2.7 An Introduction to Trees – Video 4 CART in R", I learned that to make the decision tree that based on the *Gini Index*, I need to install the rpart package.

```
> install.packages("rpart")
trying URL 'https://cran.irsn.fr/bin/macosx/contrib/4.1/rpart_4.1-15.tgz'
Content type 'application/x-gzip' length 760591 bytes (742 KB)
==================================================
downloaded 742 KB


The downloaded binary packages are in
    /var/folders/21/6jktflrn6_33bfrvbvjx3x7r0000gn/T//RtmpKAMWKg/downloaded_packages
> library(rpart)
```

I will use the BILI risk data set to process the decision tree. The train will be the first 300 rows of data set.

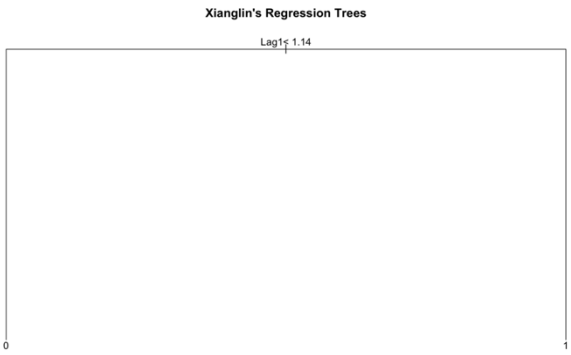
```
> Train=BILIriskNew[1:300,]
```

The usage of the rpart() is

```
rpart(formula, data, weights, subset, na.action = na.rpart, method,
      model = FALSE, x = FALSE, y = TRUE, parms, control, cost, ...)
```

I put in the train into the argument and shew the result by using plot. The summary of the rpart() seems unusual to what I researched. The Lag1 became the main variable to class the Risk Value. It probably because the Lag1 is the best variable (same as the boosting method in the A1) to class the risk.

```
> biliCART=rpart(Risk~.,data=Train,method="anova")
> biliCART
n= 300

node), split, n, deviance, yval
      * denotes terminal node

1) root 300 48 0.2
  2) Lag1< 1.14 240  0 0.0 *
  3) Lag1>=1.14 60  0 1.0 *
```



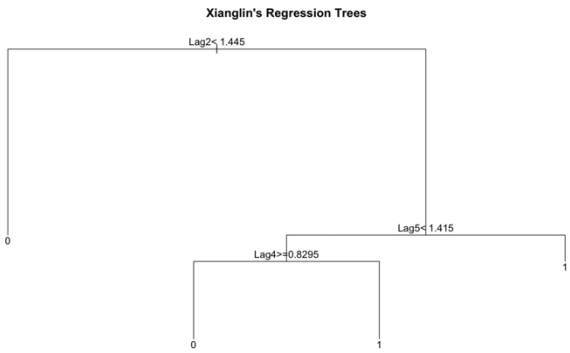
I use the model to process with predict () test set and then calculate the accuracy rate which is obviously 1.

```
> Predict=predict(biliCART,newdata=Test,type="class")
> table=table(Test$TRisk,Predict)
> table
   Predict
      0   1
  0 169   0
  1   0 352
> (table[1,1]+table[2,2])/sum(table)
[1] 1
```

So, I tried to not include the Lag1 column as A1.

```
> Train=BILIriskNew[1:300,2:11]
> head(Train)
  Lag2 Lag3 Lag4 Lag5  Lag6  Lag7  Lag8  Lag9 Lag10 TRisk
1 0.56 0.34 0.73 0.18 0.299 0.530 0.709 0.590 1.150     0
2 0.41 0.56 0.34 0.73 0.180 0.299 0.530 0.709 0.590     0
3 0.37 0.41 0.56 0.34 0.730 0.180 0.299 0.530 0.709     0
4 0.17 0.37 0.41 0.56 0.340 0.730 0.180 0.299 0.530     0
5 0.24 0.17 0.37 0.41 0.560 0.340 0.730 0.180 0.299     0
6 0.21 0.24 0.17 0.37 0.410 0.560 0.340 0.730 0.180     0
```

```
> biliCART=rpart(TRisk~.,data=Train,method="class")
> plot(biliCART,main="Xianglin's Regression Trees")
> text(biliCART)
> biliCART
n= 300

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 300 60 0 (0.8000000 0.2000000)
   2) Lag2< 1.445 268 38 0 (0.8582090 0.1417910) *
   3) Lag2>=1.445 32 10 1 (0.3125000 0.6875000)
     6) Lag5< 1.415001 20 10 0 (0.5000000 0.5000000)
      12) Lag4>=0.829499 11  3 0 (0.7272727 0.2727273) *
      13) Lag4< 0.829499 9  2 1 (0.2222222 0.7777778) *
     7) Lag5>=1.415001 12  0 1 (0.0000000 1.0000000) *
```



The classification seems more detailed. So, I tried to use this to predict the test set. I also calculate the accuracy rate, which is 0.7965.

```
> Predict=predict(biliCART,newdata=Test,type="class")
> table=table(Test$TRisk,Predict)
> table
   Predict
      0   1
  0 159  10
  1  96 256
> (table[1,1]+table[2,2])/sum(table)
[1] 0.7965451
```

Above is the practice of CART(Classification and Regression Tree) that I studied from Internet. It is based on the *Gini index* and it did has a high accuracy to predict the test set.

A3.

In A3, I will create a classification space plot using the methodology of boosting which is in A1 to plot the graph. The term classification space plot means that setting the train values as the grid and the test values as the points to visually see the classification.

I processed the BILI risk data set with the function that mentioned in the Lecture notes 8 Section 7 which is the ProbeGlm function.

I create a prove grid in R. I also changed the names of the variables to Lag1 and Lag2.

```
> x=seq(-2,7,by=0.1)
> y=seq(-2,7,by=0.1)
> StPrX=expand.grid(x,y)
> head(StPrX)
  Var1 Var2
1 -2.0   -2
2 -1.9   -2
3 -1.8   -2
4 -1.7   -2
5 -1.6   -2
6 -1.5   -2
> names(StPrX)[1]="Lag1"
> names(StPrX)[2]="Lag2"
> names(StPrX)
[1] "Lag1" "Lag2"
```

Then, I tried to create a ProbeBoost() function to plot the classification-space plot. This function is used for the boost data for A1. This function is similar to the ProbeGlm() function in lecture notes 8. I tried to fit a model using only Lag1 and Lag2.

I redo the boost.bili function to get new predictions by using the gbm() function.

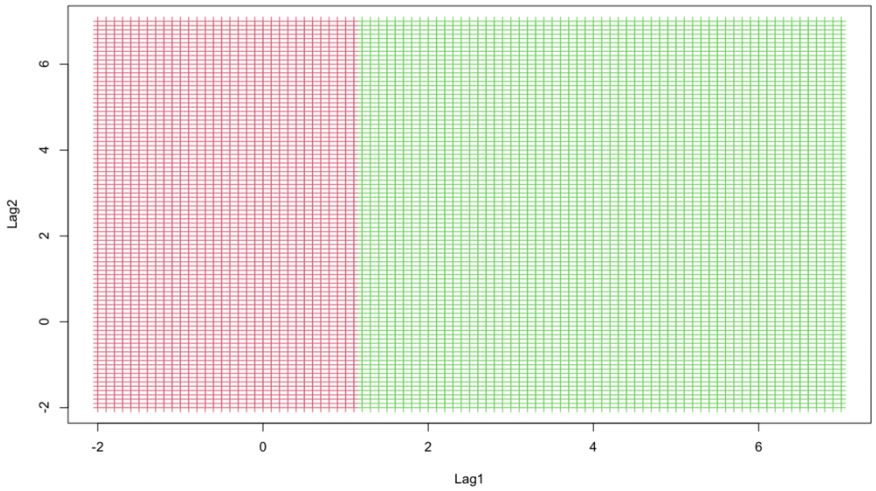
```
>
boost.bili=gbm(Risk~Lag1+Lag2,data=BILIriskNew,distribution="gaussian",n.tree=8
000,interaction.depth=5)
> yhat.bili=predict(boost.bili,newdata=StPrX,n.tree=8000)
```

I went through the data that we need for the function. The yhat.bili is the predict data set. I set up a variable called Byhat to record the Risk in yhat.bili. When the yhat.bili > -0.013 which is the median, then it is High Risk(1); on the opposite side, it will be Low Risk(0).

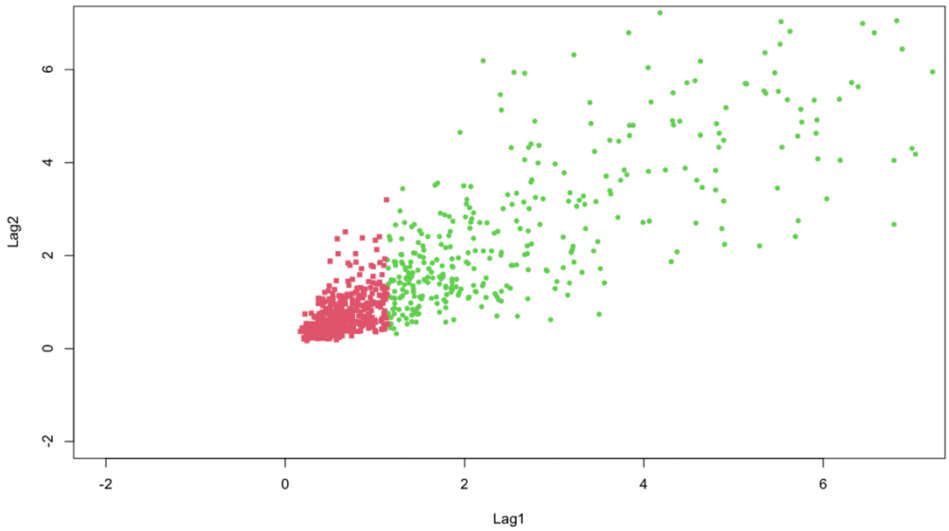
```
> median(yhat.bili)
[1] 0.9999943
> Byhat[yhat.bili>0.99]=1
> Byhat[yhat.bili<=0.99]=0
```

Since I don't know how to start, I decided to process each plot() step by step, and then combine them into a function like ProbeGlm(). When I plot the StPrX and Byhat, it looks like following.

```
> plot(StPrX,cex=1,pch=3,col=c(c(Byhat)+2),xlim=c(-2,7),ylim=c(-2,7))
```



Then I tried to plot the original x values and original y values. It looks has the same division line at Lag1=0.99.

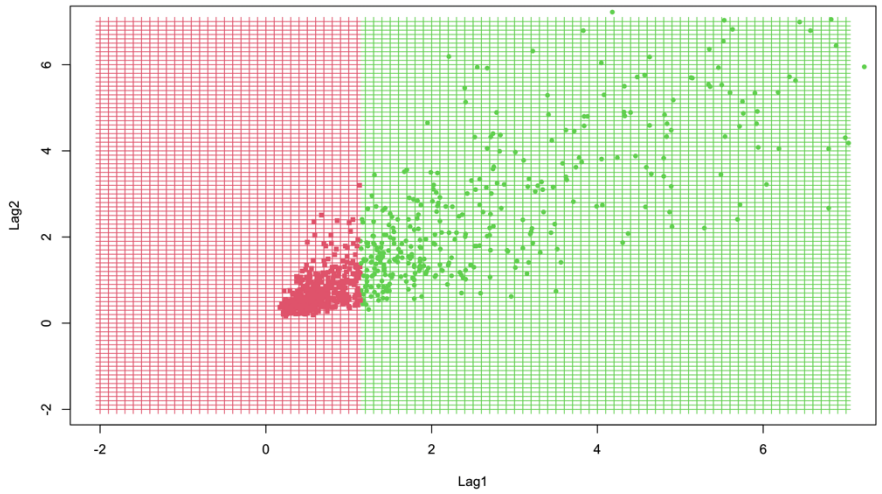



```
> plot(BILIriskNew[,c(1,2)],pch=c(BILIriskNew[,c(11)])
+15,col=c(BILIriskNew[,c(11)]+2),xlim=c(-2,7),ylim=c(-2,7),cex=0.8)
```

It seems right for the graph because the Lag1 is the most reliable variable which takes large proportion overall. So, I decided to combine them as a function.

```
> ProbeGlm = function(StPrX,Byhat,original,xlim=c(-2,7),ylim=c(-2,7))
+ {
+ plot(StPrX,cex=1,pch=3,col=c(c(Byhat)+2),xlim=xlim,ylim=ylim)
+ par(new=TRUE)
+ plot(original[,c(1,2)],pch=c(original[,c(11)])+15,col=c(original[,c(11)]
+2),xlim=xlim,ylim=ylim,cex=0.8)
+ }
> ProbeGlm(StPrX,Byhat,BILIriskNew,xlim=c(-2,7),ylim=c(-2,7))
```

```
> ProbeGlm = function(StPrX,Byhat,original,xlim=c(-2,7),ylim=c(-2,7))
+ {
+ plot(StPrX,cex=1,pch=3,col=c(c(Byhat)+2),xlim=xlim,ylim=ylim)
+ par(new=TRUE)
+
plot(original[,c(1,2)],pch=c(original[,c(11)])+15,col=c(original[,c(11)]+2),xlim=xlim,ylim=ylim,
cex=0.8)
+ }
```



According to the function, I can know that the red plot means the Low-Risk and the green plot means the High-Risk plot. However, in above graph, since the Lag1 is more related to the risk range (Lag1's index is much higher), the boundary line is a vertical line. When Lag1 is about 0.99, the predict test is being divided into different ranges.

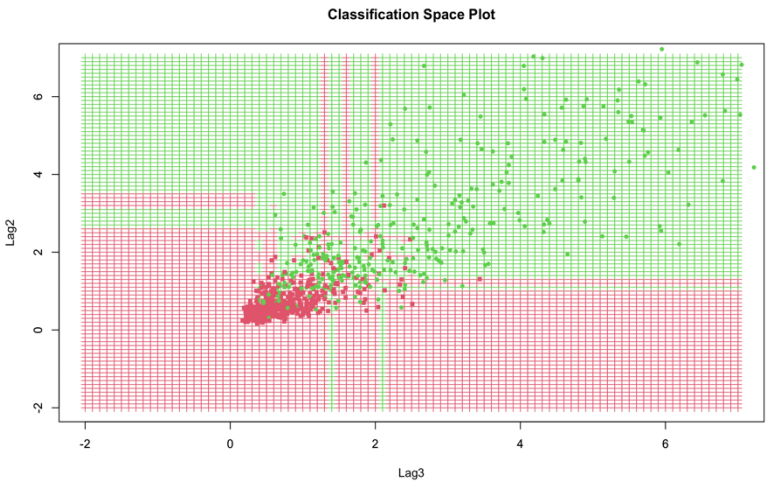
I was not satisfied with the result. Therefore, same as A1, I tried to do the plot without Lag1. I would like to see if the dividing line can be titled. Also, I want to test if the function works. At this time, I will choose the other two variables. In A1, I have tested the Lag3 and Lag2 has large portion. So, I will use Lag2 and Lag3 to see the difference. BILIsecond test is the data set without Lag1.

```
>
boost.bili=gbm(Risk~Lag3+Lag2,data=BILIsecondtest,distribution="gaussian",n.tree=8000,inte
raction.depth=5)
> names(StPrX)[1]="Lag3"
> names(StPrX)[2]="Lag2"
>  yhat.bili=predict(boost.bili,newdata=StPrX,n.tree=8000)
> median(yhat.bili)
[1] 0.6992629
> Byhat[yhat.bili>0.70]=1
> Byhat[yhat.bili<=0.70]=0
```

I changed some number and added a title to the ProbeGlm function.

```
> ProbeGlm = function(StPrX,Byhat,original,xlim=c(-2,7),ylim=c(-2,7))
{
+ plot(StPrX,cex=1,pch=3,col=c(c(Byhat)
+2),xlim=xlim,ylim=ylim,main="Classification Space Plot")
+ par(new=TRUE)
+ plot(original[,c(3,2)],ann=FALSE,pch=c(original[,c(1)])
+15,col=c(original[,c(1)]
+2),xlim=xlim,ylim=ylim,cex=0.8)
}
```

```
> ProbeGlm = function(StPrX,Byhat,original,xlim=c(-2,7),ylim=c(-2,7))
+ {
+ plot(StPrX,cex=1,pch=3,col=c(c(Byhat)+2),xlim=xlim,ylim=ylim,main="Classification Space
Plot")
+ par(new=TRUE)
+
plot(original[,c(3,2)],ann=FALSE,pch=c(original[,c(1)])+15,col=c(original[,c(1)]+2),xlim=xlim,
ylim=ylim,cex=1)
+ }
> ProbeGlm(StPrX,Byhat,BILIsecondtest,xlim=c(-2,7),ylim=c(-2,7))
```



Although I tried again for the Lag2 and Lag3, the classification space plot still looks a little bit strange since the breaking lines is not a curve or a straight line. But it is better than the previous one. And we can see the relationship between Lag2, Lag3 and division of risk range.

Part B

In the part B, I will select between 8 and 12 variables from the Cardiac data set, which is the NIH data set in LN8. Then, I will do the comparative study of performance of knn, native Bayes, logic regression (with bestglm) and boosting method on the A1.

I loaded the data set as Cardiac and then chose 10 variables from the data set to make a new data set. I created a new y-variable which is based on "mbp". I changed the continuous variable to 0-1 by mapping the values less than or equal to 0 and the rest equal to 1. Then, I appended the new mbp into the data set. The result is shown below.

```
> setwd("/Users/xianglinchen/Desktop/CIS3920/Project")
> Cardiac = read.csv("CIS-STA 3920 LN8.B Cardiac.csv")
> mbp=Cardiac[,9]
> Age=Cardiac[,13]
> median(mbp)
[1] 0.8210526
> mbp[mbp>0.8210526]=1
> mbp[mbp<=0.8210526]=0
> head(mbp)
[1] 0 1 1 0 0 1
> Cardiac1 =data.frame(Cardiac[,1:8],mbp,Age)
> Cardiac =data.frame(Cardiac[,1:8],Age,mbp)
> head(Cardiac)
        bhr    basebp    basedp      pkhr       sbp        dp     dose
1 0.5476190 0.8728814 0.4249327 0.7215190 0.3197026 0.2450142 1.333333
2 0.3690476 1.1779661 0.3864574 0.7594937 0.5873606 0.4738342 1.333333
3 0.3690476 1.1779661 0.3864574 0.7594937 0.5836431 0.4708352 1.333333
4 0.5535714 1.0000000 0.4921076 0.7468354 0.3903346 0.3096416 1.000000
5 0.5297619 0.8728814 0.4110762 0.8164557 0.6431227 0.5577298 1.333333
6 0.3452381 0.8474576 0.2600897 0.7784810 0.5204461 0.4303494 1.333333
      maxhr       Age mbp
1 0.7042254 1.2686567   0
2 0.8450704 1.0895522   1
3 0.8450704 1.0895522   1
4 0.8309859 0.8507463   0
5 0.9084507 0.5074627   0
6 0.8661972 1.0597015   1
```

KNN

The first method will be knn. I appended the kvcSearch() and kSearch() to find the prediction accuracy.

```
> kcvSearch =
+ function (X,Y,split=100, d=25, m=10)
+ {
+  if(class(Y)!= "factor") stop('Y is not factor type')
+  if(class(X)!= "data.frame") stop('X is not data.frame type')
+  rows = nrow(X)
+  Hold = rep(NA,d)
+
+  for(k in 1:d) {
+  Store = rep(NA, m)
+
+ for(j in 1:m) {
+  Shuffle=sample(rows,rows)
+  InSample=Shuffle[1:split]
+  OutSample=Shuffle[(split+1):rows]
+  TrainX = X[InSample,]
+  TrainY = Y[InSample]
+  TestX = X[OutSample,]
+  TestY = Y[OutSample]
+  knn.pred = knn(TrainX, TestX, TrainY, k)
+  table.out = table(knn.pred,TestY)
+  Store[j] = (table.out[1,1]+table.out[2,2])/sum(table.out)
+  }
+  Hold[k] = mean(Store)
+  }
+  return(Hold)
+ }
```

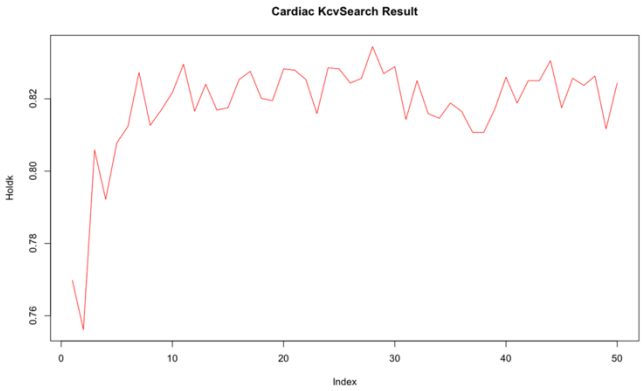
```
> kSearch =
+ function (X,Y,split=100, d=25)
+ {
+  if(class(Y)!= "factor") stop('Y is not factor type')
+  if(class(X)!= "data.frame") stop('X is not data.frame type')
+  rows = nrow(X)
+  Hold = rep(NA,d)
+  Shuffle=sample(rows,rows)
+  for(k in 1:d) {
+  InSample=Shuffle[1:split]
+  OutSample=Shuffle[(split+1):rows]
+  TrainX = X[InSample,]
+  TrainY = Y[InSample]
+  TestX = X[OutSample,]
+  TestY = Y[OutSample]
+  knn.pred = knn(TrainX, TestX, TrainY, k)
+  table.out = table(knn.pred,TestY)
+  Hold[k] = (table.out[1,1]+table.out[2,2])/sum(table.out)
+  }
+  return(Hold)
+ }
```

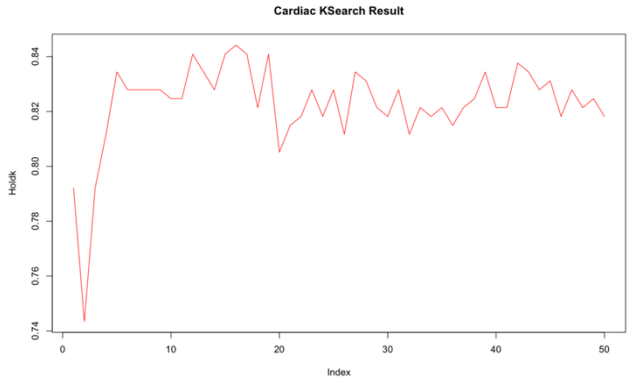
I proceed the KcvSearch(), the highest Forecasting Accuracy rate is above 0.82 with k=28.

```
> Holdk=kcvSearch(Cardiac[,1:9],as.factor(Cardiac[,10]),split=250,d=50)
> plot(Holdk,main="Cardiac KcvSearch Result",type="l",col="Red")
```

I also proceed with KSearch() which using the shuffer function. The highest Forecasting Accuracy rate is above 0.84 with k=15.

```
> Holdk=kSearch(Cardiac[,1:9],as.factor(Cardiac[,10]),split=250,d=50)
> plot(Holdk,main="Cardiac KSearch Result",type="l",col="Red")
```





Naïve Bayes

I will go through the Naïve Bayes Classification Algorithms. I used the same train data set and test data set to process the train function which uses the cross validation.

```
> CTrain.X=Cardiac[1:250,1:9]
> CTrain.Y=Cardiac[1:250,10]
> CTest.X=Cardiac[251:558,1:9]
> CTest.Y=Cardiac[251:558,10]
> CTrain.Y=as.factor(CTrain.Y)
```

The output of the train function will be called model. Then, I use the predict () to process the test data set and compare to the model. I got the accuracy rate which is about 0.88 for the Naïve Bayes model. However, in the table function, it said there's warning messages.

```
> model =train(CTrain.X,CTrain.Y,'nb',trControl=trainControl(method='cv',number=15))
> table(predict(model$finalModel, CTest.X)$class, CTest.Y)
   CTest.Y
      0   1
  0 147  24
  1  13 124
> N=table(predict(model$finalModel, CTest.X)$class, CTest.Y)
Warning message:
In FUN(X[[i]], ...) :
  Numerical 0 probability for all classes with observation 162
```

I guess it is because of the data range. Therefore, to find out why is there a warning message, I tried to uses different data set range to predict. As the range was decreasing, the warning message didn't diappared. Until I changed the range to 7:9, there's finnaly no warning messages. After I researched from internet, I undertand why. It is not because of any mistakes happened in the coding. It is more likely shows one observation is producing an unusal probabilities.

```
> CTrain.X=Cardiac[1:250,7:9]
> CTrain.Y=Cardiac[1:250,10]
> CTest.X=Cardiac[251:558,7:9]
> CTest.Y=Cardiac[251:558,10]
> CTrain.Y=as.factor(CTrain.Y)
> model =train(CTrain.X,CTrain.Y,'nb',trControl=trainControl(method='cv',number=15))
> N=table(predict(model$finalModel, CTest.X)$class, CTest.Y)
```

Solve the warning message problem, I went back to the range from 1:9 and got the accuracy rate of 0.88.

```
> N
   CTest.Y
      0   1
  0 147  24
  1  13 124
> (N[1,1]+N[2,2])/sum(N)
[1] 0.8798701
```

Logic Regression (with bestglm)

I will go through the logic regression by using the method in LN8.

First, I will process with the bestglm(). According to the LN8, I need to put in the argument of "Cardiac, IC= "BICq", "family=binomial".

```
> library(bestglm)
> best.out=bestglm(Cardiac,IC="BICq",family=binomial)
Morgan-Tatar search since family is non-gaussian.
> best.out
BICq(q = 0.25)
BICq equivalent for q in (0.000315973014261006, 0.340394691192424)
Best Model:
              Estimate Std. Error    z value     Pr(>|z|)
(Intercept) -17.345215  1.6022166 -10.825761 2.599098e-27
basebp        4.045926  0.8829624   4.582218 4.600706e-06
sbp          23.812671  2.1631779  11.008189 3.489474e-28
```

I summarized out the best.out, the "basebp" and the "sbp" will be the "best" model using the BICq criterion. Therefore, I put this two model into the glm() function as following.

```
> head(Cardiac)
       bhr    basebp    basedp      pkhr       sbp        dp     dose     maxhr       age mbp
1 0.5476190 0.8728814 0.4249327 0.7215190 0.3197026 0.2450142 1.333333 0.7042254 1.2686567   0
2 0.3690476 1.1779661 0.3864574 0.7594937 0.5873606 0.4738342 1.333333 0.8450704 1.0895522   1
3 0.3690476 1.1779661 0.3864574 0.7594937 0.5836431 0.4708352 1.333333 0.8450704 1.0895522   1
4 0.5535714 1.0000000 0.4921076 0.7468354 0.3903346 0.3096416 1.000000 0.8309859 0.8507463   0
5 0.5297619 0.8728814 0.4110762 0.8164557 0.6431227 0.5577298 1.333333 0.9084507 0.5074627   1
6 0.3452381 0.8474576 0.2600897 0.7784810 0.5204461 0.4303494 1.333333 0.8661972 1.0597015   0
> Cglm=glm(mbp~basebp+sbp,data=Cardiac, family=binomial)
> Cglm

Call:  glm(formula = mbp ~ basebp + sbp, family = binomial, data = Cardiac)

Coefficients:
(Intercept)       basebp          sbp
    -17.345        4.046       23.813

Degrees of Freedom: 557 Total (i.e. Null);  555 Residual
Null Deviance:      773.5
Residual Deviance: 359.3 AIC: 365.3
```

Then, to get the vector of estimated probabilities, I used the predict() function with the Cglm. In the function, there's only type= "response" argument in the function. The output called Cprobs at this time.

I also printed out the Cprobs with five values to check.

```
> Cprobs=predict(Cglm,type="response")
> Cprobs[1:5]
         1          2          3          4          5
0.00202405 0.80335132 0.78899038 0.01790901 0.81770378
```

I rounded the forecast to 0 or 1. I create a new variable called Cforecast. By using the value of 0.5, the Cforecast successfully changed to 0 for low mbp and 1 for high mbp.

```
> Cforecast=Cardiac$mbp
> Cforecast[1:5]
[1] 0 1 1 0 1
> Cforecast[Cprobs>0.5]=1
> Cforecast[Cprobs<0.5]=0
> summary(Cforecast)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.0000  0.0000  0.4928  1.0000  1.0000
> Cforecast[1:5]
[1] 0 1 1 0 1
> table(Cforecast,Cardiac$mbp)

Cforecast   0   1
        0 247  36
        1  34 241
> L=table(Cforecast,Cardiac$mbp)
> (L[1,1]+L[2,2])/sum(L)
[1] 0.874552
```

In this table, the correct classification rate is 0.874(87.4%) which is almost the same with the Naïve Bayes and much higher than the Knn().

My Method: Boosting Regression Tree

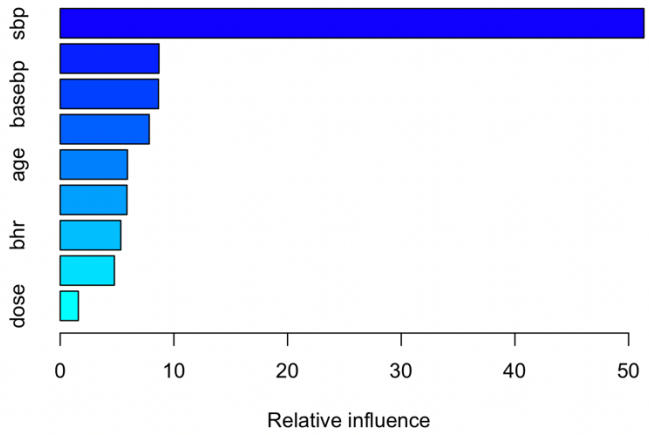
I continued with the method (gbm()) that I learned from the ISLR Textbook to see if there's better way to predict the mbp. It is the boosted regression tree.

First, I will set up a new train with a random sample.

```
> CTrain=sample(1:nrow(Cardiac),nrow(Cardiac)/2)
> library(gbm)
> set.seed(1)
> dim(Cardiac)
[1] 558  10
```

Second, I applied the Ctrain into the gbm function. I made the number of tree as 3000 and the interaction.depth as 3. From the summary, I got idea that the sbp is the must related variable to the mbp. The dp and the basebp are after the sbp having the rel.inf about 8.7.

```
> boost.cardiac = gbm(mbp~., data=Cardiac[CTrain,],distribution="gaussian",n.tree=3000,interaction.depth=3)
> summary(boost.cardiac)
          var   rel.inf
sbp       sbp 51.348601
dp         dp  8.691339
basebp basebp  8.648076
basedp basedp  7.832913
age       age  5.915022
maxhr   maxhr  5.872285
bhr       bhr  5.330493
pkhr     pkhr  4.765964
dose     dose  1.595306
```



Third, I applied the boost.cardiac model into the predict function and set up a test variable which contains the rest of data set. This model obtains a MSE (mean squared error) for 0.1058 which is very small. If I added the shrinkage parameter in the boost.cardiac, the MSE will get lower than 0.1058.

```
> Cyhat=predict(boost.cardiac,newdata=Cardiac[-CTrain,],n.tree=3000)
> Ctest=Cardiac[-CTrain,"mbp"]
> mean((Cyhat-Ctest)^2)
[1] 0.1058114
```

So, I added the shrinkage=0.2 instead of keeping the default value 0.001. At the end, I got 0.1046 as the MSE.

```
> boost.cardiac =
gbm(mbp~.,data=Cardiac[CTrain,],distribution="gaussian",n.tree=3000,interact
ion.depth=3,shrinkage=0.2,verbose=F)
> Cyhat=predict(boost.cardiac,newdata=Cardiac[-CTrain,],n.tree=3000)
> mean((Cyhat-Ctest)^2)
[1] 0.1046272
```

I also calculated the classification accuracy rate for this method. It is on the next page.

Comparative Study Conclusion

Above, I processed the Cardiac data set which has 10 variables to predict the value of mbp. At the beginning, the value of mbp has been transferred to either 1 or 0 based on if the value is over the median, 0.82. We can say that if the mbp value is 1, then the mbp is high, otherwise, it is low.

Since the Boosting Regression uses MSE (mean squared error) as the comparative value, I choose to recalculate the Accurate Classification Rate. I first convert the predict result into 1 and 0 based on if the value is above or under the average. Then, I calculated the rate by table the predict rate and true rate together. At the end, I got 0.8530466 (about 0.85).

```
> Cyhat1=Cyhat
> median(Cyhat)
[1] 0.4128135
> Cyhat1[Cyhat>0.4128135]=1
> Cyhat1[Cyhat<=0.4128135]=0
> head(Cyhat1)
[1] 1 1 1 1 0 1
> M=table(Cyhat1,Ctest)
> M
      Ctest
Cyhat1   0   1
     0 122  18
     1  23 116
> (M[1,1]+M[2,2])/sum(M)
[1] 0.8530466
```

The highest Accurate Classification Rate and lowest MSE result as list below:

Method	Measure Rate	Value
KNN	Accurate Classification Rate	0.84 with k=15
Naïve Bayes	Accurate Classification Rate	0.8798701(about 0.88)
Logic Regression (With bestglm)	Accurate Classification Rate	0.874552(about 0.87)
Boosting Regression	MSE Accurate Classification Rate	0.1046272(about 0.10) 0.8530466 (about 0.85)

From the table above, I found that the rank of the Accurate Classification rate should be: Naïve Bayes> Logic Regression (With bestglm) > Boosting Regression > KNN. Therefore, to predict the mbp, the best model is Naïve Bayes since it has a higher Accurate Classification Rate.

Appendix

8.3.1 Fitting Classification Trees

I decided to choose "classification and regression trees" (pages 330-331) to be my new classification methodology to learn in this project. Here's how I walked through the lab examples and presentation of my understanding for the contents.

Step1: Download the packages

Since "classification and regression trees" is a new methodology for calculating the risk range. It needs a new package called "trees" and new data sets called "ISLR" to download.

```
> install.packages("ISLR")
--- Please select a CRAN mirror for use in this session ---
trying URL 'https://cran.irsn.fr/bin/macosx/contrib/4.1/
ISLR_1.2.tgz'
Content type 'application/x-gzip' length 2924439 bytes (2.8 MB)
==================================================
downloaded 2.8 MB

>  install.packages("tree")
trying URL 'https://cran.irsn.fr/bin/macosx/contrib/4.1/
tree_1.0-40.tgz'
Content type 'application/x-gzip' length 178102 bytes (173 KB)
==================================================
downloaded 173 KB


The downloaded binary packages are in
   /var/folders/21/6jktflrn6_33bfrvbvjx3x7r0000gn/T//RtmpVLDvBT/
downloaded_packages
> library(tree)
```

Step2: Use classification trees to analyze the "Carseats" data set.

I processed the ifelse() function to create a new variable which records if the Sales exceed 8. When the "Yes" means the sales value exceeds 8, "No" means not. This variable called "High". Then, I make a table combine the original "Carseats" data set and the "High" variable.

```
> attach(Carseats)
> High=ifelse(Sales<=8,"No","Yes")
> Carseats=data.frame(Carseats,High)
```

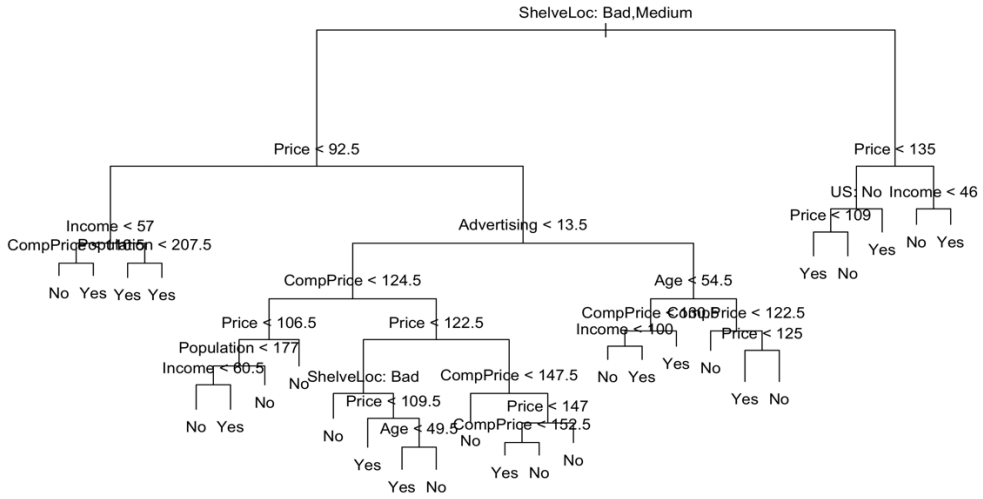
Then, I shew the summary() to list the variables that are used as internal nodes in the tree. I followed the step in the text. However, the coding in the text didn't work. It shows "Error in y - frame$yval[object$where] : non-numeric argument to binary operator" So, I checked the kind of the data. I found that the "High" variable is still formulas. I changed the "High" variable by using as.factor() so that the formula can successfully run again.

```
> tree.carseats=tree(as.factor(High)~.-Sales,Carseats)
> summary(tree.carseats)

Classification tree:
tree(formula = as.factor(High) ~ . - Sales, data = Carseats)
Variables actually used in tree construction:
[1] "ShelveLoc"   "Price"       "Income"      "CompPrice"
"Population"  "Advertising" "Age"         "US"
Number of terminal nodes:  27
Residual mean deviance:  0.4575 = 170.7 / 373
Misclassification error rate: 0.09 = 36 / 400
> class(High)
[1] "character"
```

I found that the misclassification error rate is 0.09.

I plotted the graph of tree structure and then added the text the graph for each category.

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



I used R to print out the output of the tree graph to each branch of the tree. The split criterion is the number if observations in the branch, the deviance, and overall prediction number of observations for the branch, and the fraction of observations in that branch that take on values of Yes and No.

```
> tree.carseats
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

  1) root 400 541.500 No ( 0.59000 0.41000 )
    2) ShelveLoc: Bad,Medium 315 390.600 No ( 0.68889 0.31111 )
      4) Price < 92.5 46  56.530 Yes ( 0.30435 0.69565 )
        8) Income < 57 10  12.220 No ( 0.70000 0.30000 )
```

Step3: Estimate the test error

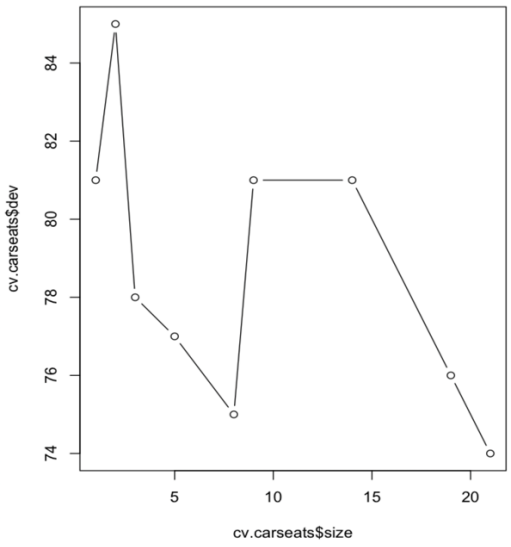
I split the observations into a training set and a test set. I used the training set as the tree and evaluate the testing set on the performance.

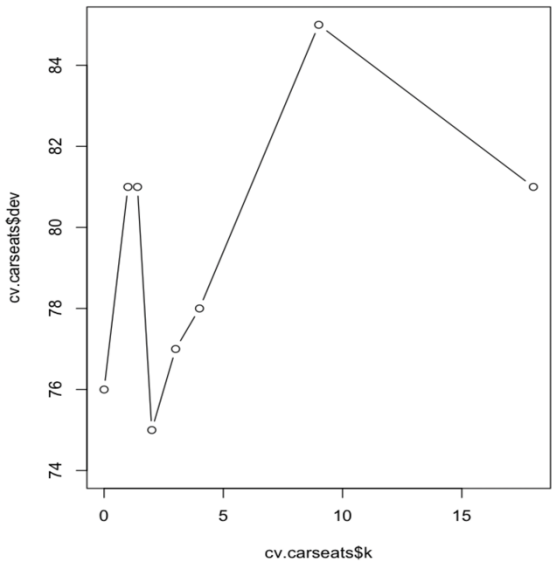
```
> set.seed(2)
> train=sample(1:nrow(Carseats),200)
> Carseats.test=Carseats[-train,]
> High.test=High[-train]
> tree.carseats=tree(as.factor(High)~.-Sales,Carseats,subset=train)
> tree.pred=predict(tree.carseats,Carseats.test,type="class")
> table(tree.pred,High.test)
         High.test
tree.pred  No Yes
      No  102  30
      Yes  15  53
> (102+53)/200
[1] 0.775
```

The correct prediction is approximately around 77.5% of the locations in the test data set. I use the cv.tree() to determine the optimal level of tree complexity. The FUN=prune.misclass can indicate that I want to use classification error to guide the cross-validation and pruning process.

```
>  set.seed(3)
> cv.carseats=cv.tree(tree.carseats, FUN=prune.misclass)
> names(cv.carseats)
[1] "size"   "dev"    "k"      "method"
> cv.carseats
$size
[1] 21 19 14  9  8  5  3  2  1

$dev
[1] 74 76 81 81 75 77 78 85 81

$k
[1] -Inf  0.0  1.0  1.4  2.0  3.0  4.0  9.0 18.0

$method
[1] "misclass"

attr(,"class")
[1] "prune"         "tree.sequence"
```

The following is the function of both size and k.

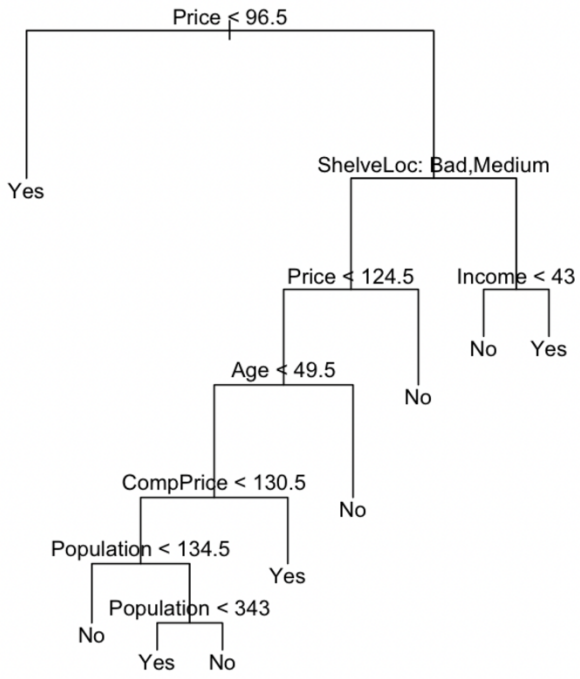





```
> par(mfrow=c(1,2))
> plot(cv.carseats$size, cv.carseats$dev,type="b")
> plot(cv.carseats$k, cv.carseats$dev,type="b")
```

I apply the prune.misclass() function in order to prune the tree to obtain the nine-node tree which results the lowest cross-validation error rate.

```
> prune.carseats=prune.misclass(tree.carseats,best=9)
> plot(prune.carseats)
> text(prune.carseats,pretty=0)
```

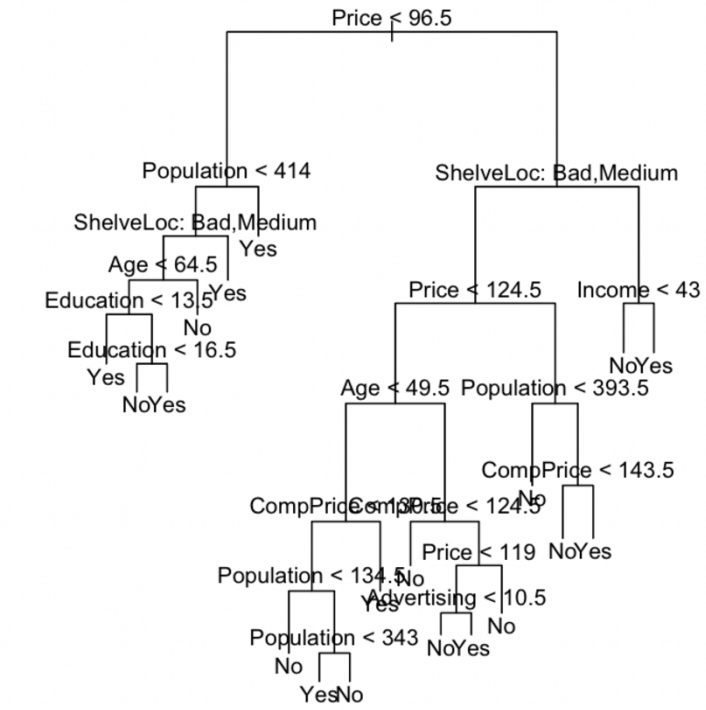


I tested the accuracy rate to prove if the graph preforms best on the test data set. I got 77.5% a little bit higher than the previous one.

```
> tree.pred=predict(prune.carseats,Carseats.test,type="class")
> table(tree.pred,High.test)
         High.test
tree.pred No Yes
      No  97  25
      Yes 20  58
> (97+58)/200
[1] 0.775
```

I also tried the prune.misclass() function to prune the tree to obtain 15-node tree.

```
> prune.carseats=prune.misclass(tree.carseats,best=15)
> plot(prune.carseats)
> text(prune.carseats,pretty=0)
```



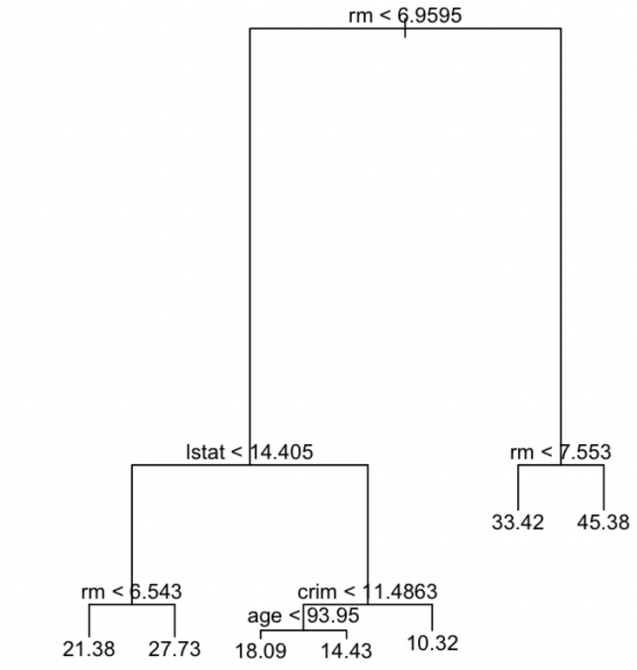
However, when I applied higher number of nodes, the classification accuracy is lower than before.

```
> tree.pred=predict(prune.carseats,Carseats.test,type="class")
> table(tree.pred,High.test)
         High.test
tree.pred  No Yes
      No  102  30
      Yes  15  53
> (102+51)/200
[1] 0.765
```

8.3.2 Fitting Regression Trees

I made a regression tree to the Boston data set.

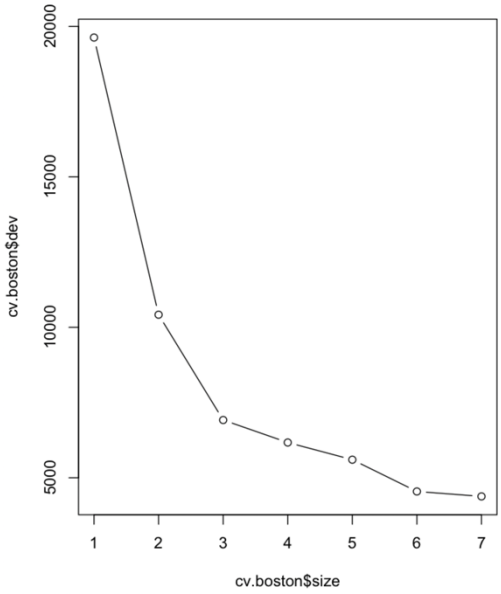
```
> library(MASS)
> set.seed(1)
> train=sample(1:nrow(Boston),nrow(Boston)/2)
> tree.boston=tree(medv~.,Boston,subset=train)
> summary(tree.boston)

Regression tree:
tree(formula = medv ~ ., data = Boston, subset = train)
Variables actually used in tree construction:
[1] "rm"    "lstat" "crim"  "age"
Number of terminal nodes:  7
Residual mean deviance:  10.38 = 2555 / 246
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-10.1800  -1.7770  -0.1775   0.0000   1.9230  16.5800

> plot(tree.boston)
> text(tree.boston,pretty=0)
```



There are 3 variables in the table to construct the tree.

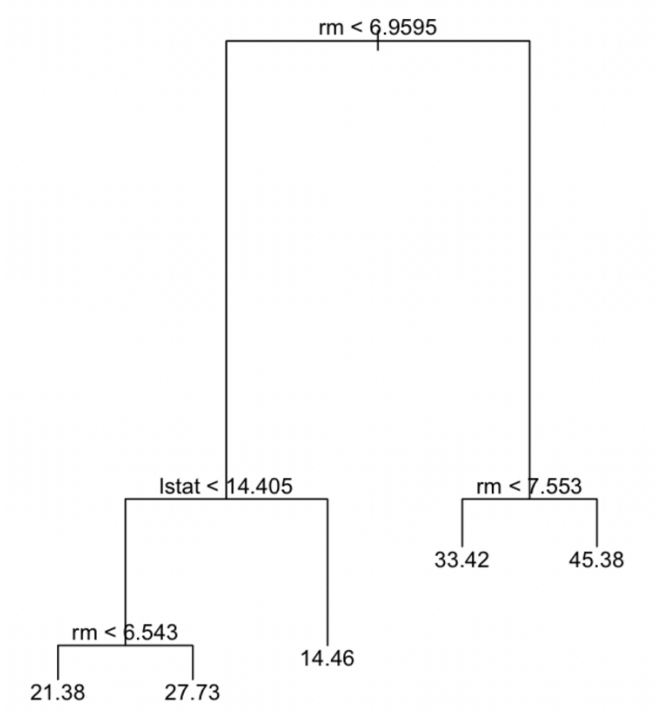
In order to improve the performance, I processed the cv.tree() function.

```
> cv.boston=cv.tree(tree.boston)
> plot(cv.boston$size,cv.boston$dev)
> plot(cv.boston$size,cv.boston$dev,type="b")
```



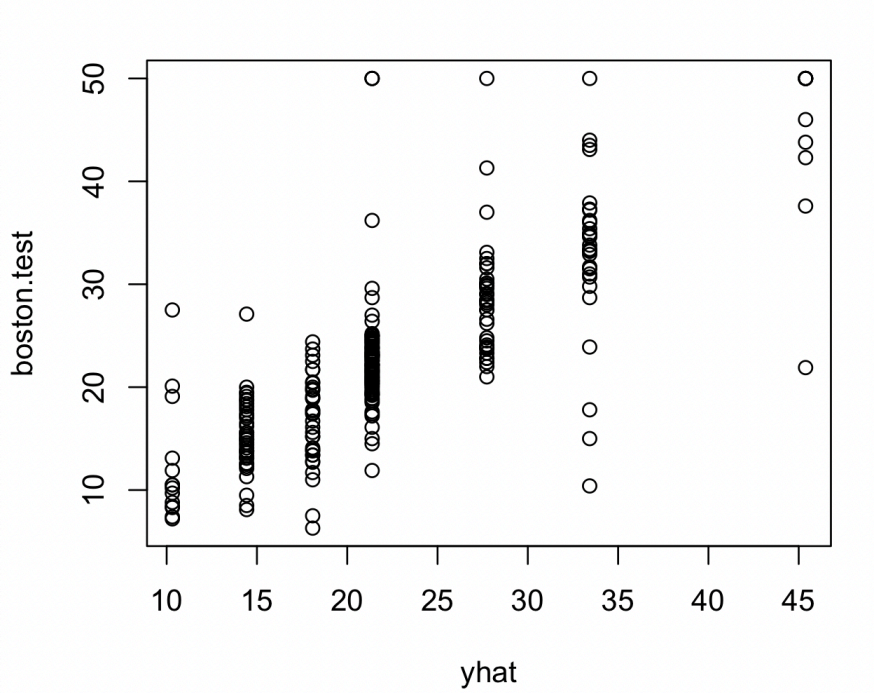
I plot the graph with variable of size and the dev.

```
> prune.boston=prune.tree(tree.boston,best=5)
> plot(prune.boston)
> text(prune.boston,pretty=0)
```



I use the unpruned tree to make predictions on the test set. I found the test set MSE associated with the regression tree is 35.05. The square root of MSE is 5.9402 which can transfer $5794.2 of the true median home value for the suburb.

```
> yhat=predict(tree.boston,newdata=Boston[-train,])
> boston.test=Boston[-train,"medv"]
> plot(yhat,boston.test)
> abline(0,1)
> mean((yhat-boston.test)^2)
[1] 35.28688
```



8.3.3 Bagging and Random Forests

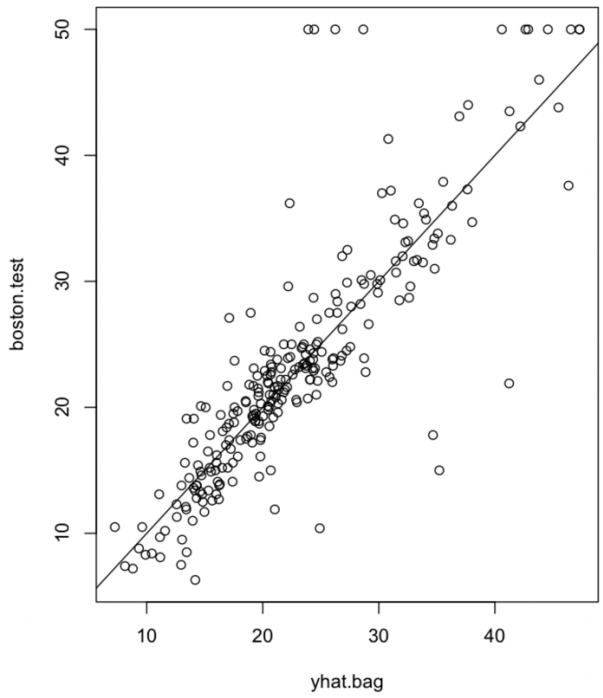
I will apply bagging and random forests to the Boston data by using the randomForest package.

I firstly download the randomForest packages and process library() in R.

```
> install.packages("randomForest")
trying URL 'https://cran.irsn.fr/bin/macosx/contrib/4.1/
randomForest_4.6-14.tgz'
Content type 'application/x-gzip' length 250573 bytes (244 KB)
==================================================
downloaded 244 KB


The downloaded binary packages are in
   /var/folders/21/6jktflrn6_33bfrvbvjx3x7r0000gn/T//RtmpVLDvBT/
downloaded_packages
> library(randomForest)
```

I will processed the prune.tree() function to make prediction ion the test stt.

```
> set.seed(1)
> bag.boston=randomForest(medv~.,data=Boston,subset=train,mtry=13,importance=TRUE)
> bag.boston

Call:
 randomForest(formula = medv ~ ., data = Boston, mtry = 13, importance = TRUE,
subset = train)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 13

          Mean of squared residuals: 11.39601
                    % Var explained: 85.17
> yhat.bag=predict(bag.boston,newdata=Boston[-train,])
> plot(yhat.bag,boston.test)
> abline(0,1)
> mean((yhat.bag-boston.test)^2)
[1] 23.59273
```



The MSE with the bagged regression tree is 23.59. I also changed the number of trees grown by randomForest() using the ntree argument

```
> bag.boston=randomForest(medv~.,data=Boston,
subset=train,mtry=13,ntree=25)
> yhat.bag=predict(bag.boston,newdata=Boston[-train,])
> mean((yhat.bag-boston.test)^2)
[1] 23.66716
```

To build a random forest of classification trees, I changed the mtry to 6 due to the default that randomForest use square root of p in the argument. The arguments resulted the test set MSE is 18.986 which shows an improvement over bagging in this case.

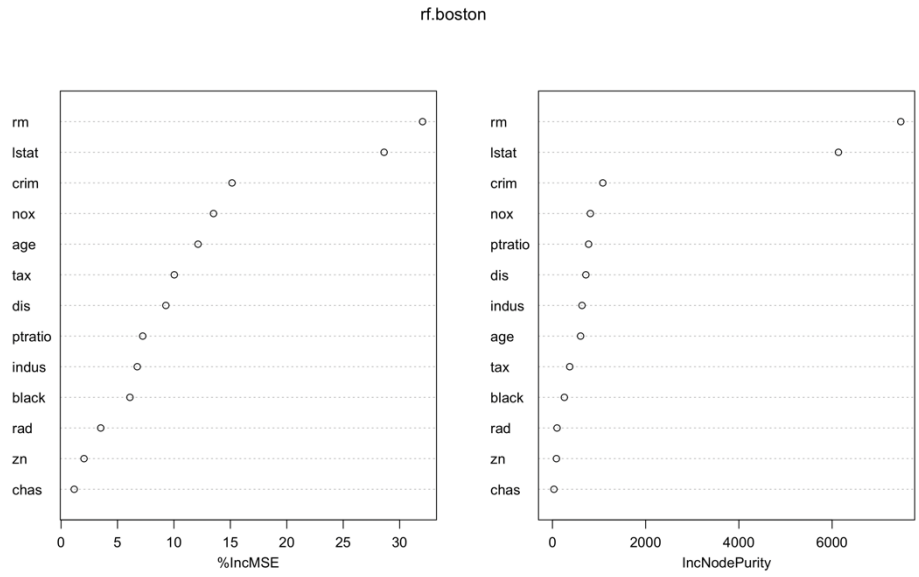
```
> rf.boston=randomForest(medv~.,data=Boston,
subset=train,mtry=6,importance=TRUE)
> yhat.rf=predict(rf.boston,newdata=Boston[-train,])
> mean((yhat.rf-boston.test)^2)
[1] 18.986
```

By using the importance() function, I can know the importance of each variable.

```
> importance(rf.boston)
          %IncMSE IncNodePurity
crim    15.147835    1080.77092
zn       2.034638      85.02213
indus    6.749944     634.60426
chas     1.165267      32.28978
nox     13.506885     813.23301
rm      32.029666    7473.59568
age     12.134563     603.62403
dis      9.285196     717.09762
rad      3.514649      98.17352
tax     10.031711     370.24996
ptratio  7.235556     775.66024
black    6.100979     255.50450
lstat   28.622141    6135.66690
```

Then I plot of these importance measures using the varImpPlot() function.

```
> varImpPlot(rf.boston)
```



8.3.4 Boosting

To process the boosted regression trees to the Boston data set, I downloaded the gbm package within the gbm() function.

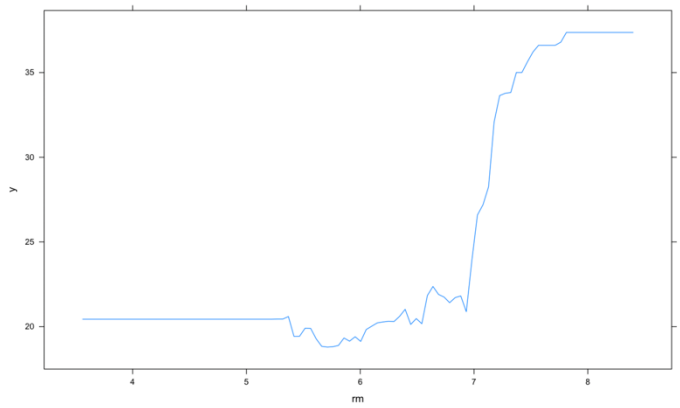
```
> install.packages("gbm")
trying URL 'https://cran.irsn.fr/bin/macosx/contrib/4.1/gbm_2.1.8.tgz'
Content type 'application/x-gzip' length 1025250 bytes (1001 KB)
==================================================
downloaded 1001 KB


The downloaded binary packages are in
   /var/folders/21/6jktflrn6_33bfrvbvjx3x7r0000gn/T//RtmpVLDvBT/downloaded_packages
> library("gbm")
Loaded gbm 2.1.8
```

Since the purpose of problem is to build a regression, I use distribution= “gaussian” in this argument.

```
> set.seed(1)
> boost.boston=gbm(medv~.,data=Boston[train,],distribution="gaussian",n.trees=5000,interaction.depth=4)
> summary(boost.boston)
            var    rel.inf
rm           rm 43.9919329
lstat     lstat 33.1216941
crim       crim  4.2604167
dis         dis  4.0111090
nox         nox  3.4353017
black     black  2.8267554
age         age  2.6113938
ptratio ptratio  2.5403035
tax         tax  1.4565654
indus     indus  0.8008740
rad         rad  0.6546400
zn           zn  0.1446149
chas       chas  0.1443986
```

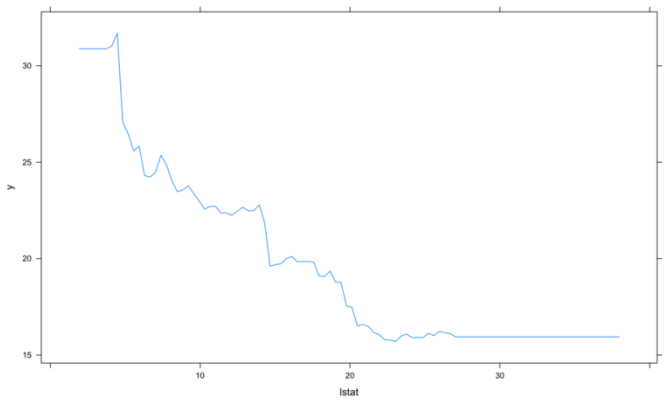
I plot the partial dependence plots for the two variables.

```
> par(mfrow=c(1,2))
> plot(boost.boston,i="rm")
```



For the rm, median house prices are increasing with rm.

```
> plot(boost.boston,i="lstat")
```



For the lstat, median house prices are decreasing with rm.

```
> yhat.boost=predict(boost.boston,newdata=Boston[-train,],
n.trees=5000)
> mean((yhat.boost-boston.test)^2)
[1] 18.84709
```

The test MSE is 18.16 which is similar to the random trees. I then changed the default value of shrinkage parameter to 0.2.

```
>
boost.boston=gbm(medv~.,data=Boston[train,],distribution="gaussian",n.tree=5000,interaction.depth=4,shri
nkage=0.2,verbose=F)
> yhat.boost=predict(boost.boston,newdata=Boston[-train,],n.tree=5000)
> mean((yhat.boost-boston.test)^2)
[1] 18.15816
```

When using 0.2 as the shrinkage, it leads lower test MSE than 0.001.

Works Cited

-Cover Page Picture

"Chinese Audio and Video Creation Platform Bilibili Experiences Temporary Inaccessibility Due to Engine Room Failure". *Pandaily.* July 14, 2021. https://pandaily.com/chinese-audio-and-video-creation-platform-bilibili-experiences-temporary-inaccessibility-due-to-engine-room-failure/.

-A2

Bui, Huy. "Decision Tree Fundamentals". *Towards Data Science.* Mar 31, 2020. https://towardsdatascience.com/decision-tree-fundamentals-388f57a60d2a.

"4.2.7 An Introduction to Trees – Video 4 CART in R". *Youtube*, uploaded by MIT OpenCourseWare. Dec 13, 2018. https://www.youtube.com/watch?v=JvtqThS69bw.

-Text Book

Gareth James, Daniela Witten, and Trevor Hastie. An Introduction to Statistical Learning with applications in R. Springer Science + Business median New York, 2013.