Data Science & Analytics ASSIGNMENT

## SHIVAANI KATRAGADDA

## R00183214

#Part 2- f,g,h questions

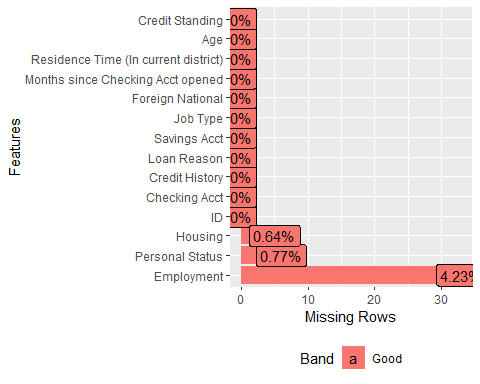
# install.packages("readxl")  
# install.packages("DataExplorer")  
# install.packages("rpart")  
# install.packages("C50")  
# install.packages("caret")  
  
  
  
library(readxl)  
library(DataExplorer)  
library(rpart)  
library(C50)  
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

#Here i am using this packages to read the files of given dataset and   
#to build the decision tree model so that i can plot the roc plot by using the predicted values and actual values of the models  
#readxl for reading excel sheet,DataExplorer to see the missing values of data,rpart for building decision tree,caret for confusion matrix for decision tree and boosting algorithm  
#and c50 for boosting algorithm(this is giving good acuracy among all the other)

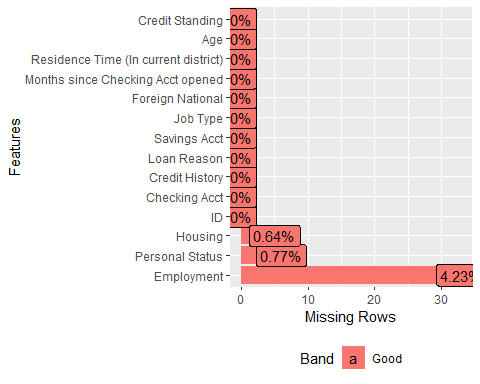
#Reading the scoring\_Data of Credit\_Risk6\_final.xlsx file by using read\_excel() function and it is assigned to Credit\_Risk6\_final1  
  
Credit\_Risk6\_final1 <- read\_excel("F:/Ds Assignment/Credit\_Risk6\_final.xlsx",sheet = "Scoring\_Data")  
  
#Generating the dataframe for Credit\_Risk6\_final1 using as.data.frame() function and that dataframe is named as Credit\_Scoring\_Data   
Credit\_Scoring\_Data<- as.data.frame(Credit\_Risk6\_final1)  
  
#Reading the Training\_Data of Credit\_Risk6\_final.xlsx file by using read\_excel() function and it is assigned to Credit\_Risk6\_final2  
Credit\_Risk6\_final2 <- read\_excel("F:/Ds Assignment/Credit\_Risk6\_final.xlsx", sheet = "Training\_Data")  
  
#Generating the dataframe for Credit\_Risk6\_final2 using as.data.frame() function and that dataframe is named as Credit\_Training\_Data  
Credit\_Training\_Data<- as.data.frame(Credit\_Risk6\_final2)  
  
#plot\_missing() function shows the percentage of missing values of each column present in the dataset  
plot\_missing(Credit\_Training\_Data)



#Filling the NA values  
  
#The missing data is a categorical data so i am replacing the missing values with the mode   
  
dim(Credit\_Training\_Data) #dim() will give the number of rows and columns in the dataset

## [1] 780 14

#plot\_missing() function shows the percentage of missing values of each column present in the dataset  
plot\_missing(Credit\_Training\_Data)



#from plot\_missing() we know that we have missing values in three columns they are Housing,Personal status and Employment.  
  
table(is.na(Credit\_Training\_Data$Housing))#checking how many misssig values are there in the column

##   
## FALSE TRUE   
## 775 5

sort(table(Credit\_Training\_Data$Housing))#sorting the column in order to get the total number of observation for each domain in the column in ascending order

##   
## Other Rent Own   
## 94 157 524

names(table(Credit\_Training\_Data$Housing))[table(Credit\_Training\_Data$Housing)==max(table(Credit\_Training\_Data$Housing))]#now taking the domain which is having the high value

## [1] "Own"

Credit\_Training\_Data$Housing[is.na(Credit\_Training\_Data$Housing)] <- "Own"#Now assigining the highest value to the missing rows in the column  
table(is.na(Credit\_Training\_Data$Housing))#checking whether all the missing values are replace by running this line again

##   
## FALSE   
## 780

table(is.na(Credit\_Training\_Data$Employment))#checking how many misssig values are there in the column

##   
## FALSE TRUE   
## 747 33

sort(table(Credit\_Training\_Data$Employment))#sorting the column in order to get the total number of observation for each domain in the column in ascending order

##   
## Retired Unemployed Very Short Medium Long Short   
## 2 43 134 140 186 242

names(table(Credit\_Training\_Data$Employment))[table(Credit\_Training\_Data$Employment)==max(table(Credit\_Training\_Data$Employment))] #now taking the domain which is having the high value

## [1] "Short"

Credit\_Training\_Data$Employment[is.na(Credit\_Training\_Data$Employment)] <- "Short"#Now assigining the highest value to the missing rows in the column  
table(is.na(Credit\_Training\_Data$Employment))#checking whether all the missing values are replace by running this line again

##   
## FALSE   
## 780

table(is.na(Credit\_Training\_Data$`Personal Status`))#checking how many misssig values are there in the column

##   
## FALSE TRUE   
## 774 6

sort(table(Credit\_Training\_Data$`Personal Status`))#sorting the column in order to get the total number of observation for each domain in the column in ascending order

##   
## Married Divorced Single   
## 70 273 431

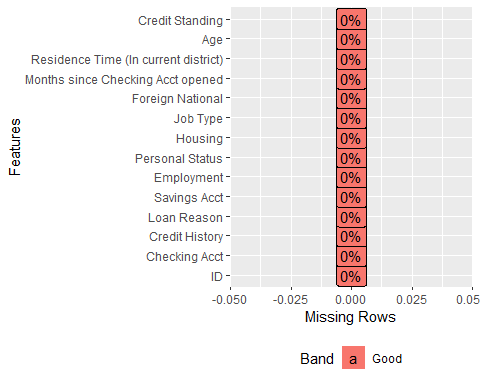
names(table(Credit\_Training\_Data$`Personal Status`))[table(Credit\_Training\_Data$`Personal Status`)==max(table(Credit\_Training\_Data$`Personal Status`))]#now taking the domain which is having the high value

## [1] "Single"

Credit\_Training\_Data$`Personal Status`[is.na(Credit\_Training\_Data$`Personal Status`)] <- "Single"#Now assigining the highest value to the missing rows in the column  
table(is.na(Credit\_Training\_Data$`Personal Status`))#checking whether all the missing values are replace by running this line again

##   
## FALSE   
## 780

#All the missing data is imputated with mode check again whether any missimg data is present in the datase by using plot\_missing()  
plot\_missing(Credit\_Training\_Data)



### QUESTION(f)

Develop a InfoGain algorithm that works on this dataset to calculate the variable for the first split. You may use the code developed in the labs as a starting point but make sure to annotate your code with comments explaining what it is doing. Note you can only used base R commands here no other packages are allowed. Comment on your results.

#creating one function i.e tabfun which contains the proprtion table with all the columns in the Credit\_Training\_Data along with laplace smoothing(which allows unrepresented value to show up) with margin=1  
  
tabfun <- function(x) {prop.table(table(Credit\_Training\_Data[,x],Credit\_Training\_Data[,14]) + 1e-6, margin = 1)}   
  
# The formula for entropy is. -1 \*probability of a false \* log2( of this probability)  
  
# Now we need rowSums of this, i.e   
  
rowSums(-tabfun(5)\*log2(tabfun(5)))#checking the function by passing one column value and finding rowsums

## High Low MedHigh MedLow No Acct   
## 0.8935711 0.9866165 0.9755260 0.8960382 0.9719015

# Now bring it altogether with one formula and writing one function to find entropy i.e entopy\_tab  
  
entopy\_tab <- function(x) { tabfun <- prop.table(table(Credit\_Training\_Data[,x],Credit\_Training\_Data[,14])+ 1e-6, margin = 1)  
sum(prop.table(table(Credit\_Training\_Data[,x]))\*rowSums(-tabfun(x)\*log2(tabfun(x))))}  
  
#Here I am writiong one for loop for finding the entopy value for the columns 2 to 13  
for (i in 2:13){  
 print(colnames(Credit\_Training\_Data[i]))#printing the each column name of credit\_Training\_Data  
 entropy = entopy\_tab(i)#finding the entropy for each column by calling enrtopy\_tab function and storing the result in the variable entropy  
 print (entropy)#printing the value of entopy  
}

## [1] "Checking Acct"  
## [1] 0.950357  
## [1] "Credit History"  
## [1] 0.7179562  
## [1] "Loan Reason"  
## [1] 0.9596747  
## [1] "Savings Acct"  
## [1] 0.9708688  
## [1] "Employment"  
## [1] 0.9283017  
## [1] "Personal Status"  
## [1] 0.9693748  
## [1] "Housing"  
## [1] 0.9674949  
## [1] "Job Type"  
## [1] 0.9648772  
## [1] "Foreign National"  
## [1] 0.9758157  
## [1] "Months since Checking Acct opened"  
## [1] 0.9129813  
## [1] "Residence Time (In current district)"  
## [1] 0.9681393  
## [1] "Age"  
## [1] 0.8670898

Creditstanding\_prop\_table <- prop.table(table(Credit\_Training\_Data$`Credit Standing`))#making proportion table for credit standing in Crexdit\_Training\_Data and storing in Creditstanding\_prop\_table  
Creditstanding\_prop\_table#printing the Creditstanding\_prop\_table value

##   
## Bad Good   
## 0.4089744 0.5910256

entopy\_total <-sum(-Creditstanding\_prop\_table\*log2(Creditstanding\_prop\_table))#finding the entopy value Creditstanding\_prop\_table,considering it as total entopy and storing in the variable entopy\_total  
entopy\_total

## [1] 0.9759588

infogain.list<-NULL#initializing one empty list i.e infogain.list  
  
#Now writing one function to find infogain i.e infogain()  
infogain<-function(x)  
{  
 gain=entopy\_total-entopy\_tab(x)#finding the infogain and storing it in variable gain  
 return(gain)#returning the infogain value i.e gain   
}  
#Here I am writing one for loop for finding the infogain value for the columns 2 to 13  
  
for (i in 2:13){  
 print(i)#printing the value of i  
 info = infogain(i)#finding the infogain for each column by calling the function infogain() and storing the result in the variable info  
 print (info)#printig the result of info  
 infogain.list<-c(infogain.list,info)#assigining a vector which contains infogain.list,info to infogain.list   
 print(colnames(Credit\_Training\_Data[i]))#printing the each column name of credit\_Training\_Data  
  
}

## [1] 2  
## [1] 0.02560174  
## [1] "Checking Acct"  
## [1] 3  
## [1] 0.2580026  
## [1] "Credit History"  
## [1] 4  
## [1] 0.01628411  
## [1] "Loan Reason"  
## [1] 5  
## [1] 0.005090016  
## [1] "Savings Acct"  
## [1] 6  
## [1] 0.04765704  
## [1] "Employment"  
## [1] 7  
## [1] 0.006583928  
## [1] "Personal Status"  
## [1] 8  
## [1] 0.008463851  
## [1] "Housing"  
## [1] 9  
## [1] 0.01108154  
## [1] "Job Type"  
## [1] 10  
## [1] 0.0001430321  
## [1] "Foreign National"  
## [1] 11  
## [1] 0.06297746  
## [1] "Months since Checking Acct opened"  
## [1] 12  
## [1] 0.007819437  
## [1] "Residence Time (In current district)"  
## [1] 13  
## [1] 0.108869  
## [1] "Age"

infogain.list#displaying the values in the infogaion.list

## [1] 0.0256017377 0.2580026235 0.0162841145 0.0050900156 0.0476570427  
## [6] 0.0065839275 0.0084638512 0.0110815377 0.0001430321 0.0629774639  
## [11] 0.0078194368 0.1088689952

mainindex<-which.max(infogain.list)#getting the maximin index from the infogain.list and storing it in the variable mainindex  
mainindex#printing the mainindex

## [1] 2

print(colnames(Credit\_Training\_Data[mainindex+1]))#now printing the column name of the mainindex

## [1] "Credit History"

#the column Credit History is the first split

The information gain is calculated by using the formula total entropy minus entropy where entropy formula is as follows -1 *probability of a false*  log2( of this probability).

1. Initally I Created one function called tabfun which contains the proportion table with all the columns in the Credit\_Training\_Data along with Laplace smoothing(which allows unrepresented value to show up) with margin=1.
2. The formula of entropy is -1 *probability of a false*  log2( of this probability). Now I zam taking row sums by using the entropy formula.
3. Now bringing step1 and step2 together I am writing one function to find entropy i.e entopy\_tab
4. I implemented one for loop for finding entropy value for columns to 2 to 13 ,here I am not finding the entropy for first column i.e ID which is a column with unique values and It does not have much impact while creating the decision tree and I am ignoring the fourteenth column i.e credit standing which is a label variable .
5. Now finding the total entropy by using the label variable i.e credit standing column
6. I am initializing an empty list i.e infogain.list
7. I am writing one function to find information gain that is called as info gain() In that function, I am finding total entropy minus entropy to get the info gain
8. I implemented one for loop for finding info gain value for columns 2 to 13,here I am finding each column info gain and assigning that value to info variable. Later I assigned a vector that contains infogain.list and info to infogain.list.Later printing the name of each column, which will appear below the info gain value while printing the output.
9. I am finding the main index i.e first split by taking the maximum info gain value and printing the column name of that main index which will give the column name “Credit History” Among all the columns info gain value the Credit History column has the highest value so it will be the first split

Therefore “Credit History” is the first split of decision tree

### QUESTION(g)

Develop code in R that illustrates how boosting works using the formulae for adabag in the attached document. Use the Excel spreadsheet attached so that you have only ten data points. Use set.seed(abc) with abc being the last 3 digits of your student number to generate a random prediction (each time) for 4 iterations of boosting. Include a confusion matrix at the end for your final prediction and comment. Note you can only used base R commands here no other packages are allowed.

set.seed(214)#setting the seed with the last three digits of my student number  
id<-c(1,2,3,4,5,6,7,8,9,10)#Taking 10 values in a vector which are called ID's and assigining that vector to Variable ID  
label<-c(0,1,1,0,1,1,0,1,0,0)#Taking 10 values in a vector which are called label and assigining that vector to Variable Label  
weights<-c(0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1)#Taking 10 values in a vector which are called Weights and assigining that vector to Variable Weights  
  
counter <-0#initializing a counter with value zero  
currentDF<-data.frame()#initializing one empty dataframe i.e called as currentDF  
nextDF <- data.frame()#initializing one empty dataframe i.e called as nextDF  
#writing while loop for implementing adaboost  
while (counter < 4) {  
 #If counteris equal   
 if (counter == 0){  
 currentDF <- data.frame(id, label, weights)#assign is,label,weights to currentDf dataframe  
 } else {  
 currentDF <- nextDF#if counter not equal to zero assign nextDF dataframe to currentDF  
 }  
 currentDF$prediction <- sample(0:1, 10, replace=TRUE)#generating random numbers between 0 and 1 ten times and assigining that to currentDF$prediction  
 currentDF$error <- ifelse(currentDF$label==currentDF$prediction,0,1)#Now finding error value if label value is equal to prediction value error value will be 0 else error value is 1,and assigning error value to CurrentDF$error  
 currentDF$xy <- currentDF$weights\*currentDF$error#now finding the product of weights and error and assigining to currentDF$xy  
   
 sumofxy <- sum(currentDF$xy)#Finding sum of currentDF$xy  
 alpha <- 0.5 \* (log((1-sumofxy)/sumofxy))#finding alpha value by using the given formula  
 incorrect <- exp(-alpha\*-1)#finding the incorrect value by using the given formula  
 correct <- exp(-alpha\*1)#finding the correct value by using the given formula  
   
 #Now finding adjustment value if label value is equal to prediction value error value will be correct else error value is incorrect,and assigning adjustment value to CurrentDF$adjustment  
 currentDF$adjustment <- ifelse(label==currentDF$prediction,correct,incorrect)  
 #now finding the product of adjustment\*weights and assigining it to the CurrentDF$adj\_weight   
 currentDF$adj\_weight <- (currentDF$adjustment)\*(currentDF$weights)  
 #Now Finding the new\_weight value i.e adj\_weight divided by sum of adj\_weight and assigining it to currentDF$new\_weight  
 currentDF$new\_weight <- currentDF$adj\_weight/sum(currentDF$adj\_weight)  
   
 nextDF <- currentDF#Assigining all the currentDF dataframe values to nextDF dataframe  
 nextDF$weights <- currentDF$new\_weight#assigining the new\_weights value of currentDF dataframe to weights of nextDF dataframe value   
 cat("\n")  
 print(currentDF)#printing the currentDF dataframe  
   
 counter <- counter + 1#incrementing the value of counter  
}

##   
## id label weights prediction error xy adjustment adj\_weight new\_weight  
## 1 1 0 0.1 0 0 0.0 0.8164966 0.08164966 0.08333333  
## 2 2 1 0.1 1 0 0.0 0.8164966 0.08164966 0.08333333  
## 3 3 1 0.1 0 1 0.1 1.2247449 0.12247449 0.12500000  
## 4 4 0 0.1 1 1 0.1 1.2247449 0.12247449 0.12500000  
## 5 5 1 0.1 1 0 0.0 0.8164966 0.08164966 0.08333333  
## 6 6 1 0.1 1 0 0.0 0.8164966 0.08164966 0.08333333  
## 7 7 0 0.1 0 0 0.0 0.8164966 0.08164966 0.08333333  
## 8 8 1 0.1 0 1 0.1 1.2247449 0.12247449 0.12500000  
## 9 9 0 0.1 1 1 0.1 1.2247449 0.12247449 0.12500000  
## 10 10 0 0.1 0 0 0.0 0.8164966 0.08164966 0.08333333  
##   
## id label weights prediction error xy adjustment adj\_weight  
## 1 1 0 0.08333333 0 0 0.00000000 1.9493589 0.16244657  
## 2 2 1 0.08333333 0 1 0.08333333 0.5129892 0.04274910  
## 3 3 1 0.12500000 1 0 0.00000000 1.9493589 0.24366986  
## 4 4 0 0.12500000 1 1 0.12500000 0.5129892 0.06412365  
## 5 5 1 0.08333333 0 1 0.08333333 0.5129892 0.04274910  
## 6 6 1 0.08333333 0 1 0.08333333 0.5129892 0.04274910  
## 7 7 0 0.08333333 1 1 0.08333333 0.5129892 0.04274910  
## 8 8 1 0.12500000 0 1 0.12500000 0.5129892 0.06412365  
## 9 9 0 0.12500000 1 1 0.12500000 0.5129892 0.06412365  
## 10 10 0 0.08333333 1 1 0.08333333 0.5129892 0.04274910  
## new\_weight  
## 1 0.20000000  
## 2 0.05263158  
## 3 0.30000000  
## 4 0.07894737  
## 5 0.05263158  
## 6 0.05263158  
## 7 0.05263158  
## 8 0.07894737  
## 9 0.07894737  
## 10 0.05263158  
##   
## id label weights prediction error xy adjustment adj\_weight  
## 1 1 0 0.20000000 1 1 0.20000000 0.7211103 0.14422205  
## 2 2 1 0.05263158 1 0 0.00000000 1.3867505 0.07298687  
## 3 3 1 0.30000000 0 1 0.30000000 0.7211103 0.21633308  
## 4 4 0 0.07894737 0 0 0.00000000 1.3867505 0.10948030  
## 5 5 1 0.05263158 0 1 0.05263158 0.7211103 0.03795317  
## 6 6 1 0.05263158 1 0 0.00000000 1.3867505 0.07298687  
## 7 7 0 0.05263158 1 1 0.05263158 0.7211103 0.03795317  
## 8 8 1 0.07894737 1 0 0.00000000 1.3867505 0.10948030  
## 9 9 0 0.07894737 0 0 0.00000000 1.3867505 0.10948030  
## 10 10 0 0.05263158 1 1 0.05263158 0.7211103 0.03795317  
## new\_weight  
## 1 0.15200000  
## 2 0.07692308  
## 3 0.22800000  
## 4 0.11538462  
## 5 0.04000000  
## 6 0.07692308  
## 7 0.04000000  
## 8 0.11538462  
## 9 0.11538462  
## 10 0.04000000  
##   
## id label weights prediction error xy adjustment adj\_weight  
## 1 1 0 0.15200000 0 0 0.00000000 1.2657336 0.19239151  
## 2 2 1 0.07692308 1 0 0.00000000 1.2657336 0.09736413  
## 3 3 1 0.22800000 0 1 0.22800000 0.7900556 0.18013268  
## 4 4 0 0.11538462 1 1 0.11538462 0.7900556 0.09116027  
## 5 5 1 0.04000000 0 1 0.04000000 0.7900556 0.03160223  
## 6 6 1 0.07692308 0 1 0.07692308 0.7900556 0.06077351  
## 7 7 0 0.04000000 0 0 0.00000000 1.2657336 0.05062935  
## 8 8 1 0.11538462 0 1 0.11538462 0.7900556 0.09116027  
## 9 9 0 0.11538462 0 0 0.00000000 1.2657336 0.14604619  
## 10 10 0 0.04000000 1 1 0.04000000 0.7900556 0.03160223  
## new\_weight  
## 1 0.19775821  
## 2 0.10008006  
## 3 0.18515742  
## 4 0.09370315  
## 5 0.03248376  
## 6 0.06246877  
## 7 0.05204163  
## 8 0.09370315  
## 9 0.15012010  
## 10 0.03248376

print(currentDF)#Printing the final result of currentDF

## id label weights prediction error xy adjustment adj\_weight  
## 1 1 0 0.15200000 0 0 0.00000000 1.2657336 0.19239151  
## 2 2 1 0.07692308 1 0 0.00000000 1.2657336 0.09736413  
## 3 3 1 0.22800000 0 1 0.22800000 0.7900556 0.18013268  
## 4 4 0 0.11538462 1 1 0.11538462 0.7900556 0.09116027  
## 5 5 1 0.04000000 0 1 0.04000000 0.7900556 0.03160223  
## 6 6 1 0.07692308 0 1 0.07692308 0.7900556 0.06077351  
## 7 7 0 0.04000000 0 0 0.00000000 1.2657336 0.05062935  
## 8 8 1 0.11538462 0 1 0.11538462 0.7900556 0.09116027  
## 9 9 0 0.11538462 0 0 0.00000000 1.2657336 0.14604619  
## 10 10 0 0.04000000 1 1 0.04000000 0.7900556 0.03160223  
## new\_weight  
## 1 0.19775821  
## 2 0.10008006  
## 3 0.18515742  
## 4 0.09370315  
## 5 0.03248376  
## 6 0.06246877  
## 7 0.05204163  
## 8 0.09370315  
## 9 0.15012010  
## 10 0.03248376

#NOW confusion matrix  
  
table(currentDF$label,currentDF$prediction)#making the table for label and prediction of currentDF

##   
## 0 1  
## 0 3 2  
## 1 4 1

#we can find true positive and true negatives from table and Add the true positives and true negatives and then divide by all the values to find the confusion matrix  
#here the values will not change because we are using set.seed(214) so directly finding confusion matrix by using the values from the table   
cat("The confusion matrix is",(3+1)/(3+2+4+1))#finding the confusion matrix and printing the result

## The confusion matrix is 0.4

Adabag or AdaBoost (Adaptive Boosting) is another widely used boosting algorithm in machine learning. Improving week learners and creating an aggregated model to improve model accuracy is a key concept of boosting algorithms. A weak learner is defined as the one with poor performance or slightly better than a random guess classifier. Adaboost improves those classifiers by increasing their weights and gets their votes to create the final combined model.

To implement this algorithm I have given an excel sheet Boosting calcs - Lecture 6v4 which contains 10 rows and 9 columns they are as follows id, label, weights, prediction, Error, one unnamed column which is the product of weights and error, adjustments,adj\_weights, and new weights with initials values for all the columns. I will consider ID, Label, Weights columns only because in question it is given that randomly generate the values and procced the algorithm process for four iterations

1. I initially took the Id, LABEL, WEIGHTS values from the excel sheet.I initialized vector for each column and assigned to id, label, weights variables.
2. After that I generated two empty data frames i.e currentDF,nextDF.I implemented a while loop to run for four iterations.
3. if the counter is equal to zero id, label, weights columns are added to the currentDF if counter not equal to zero I assigned nextDF data frame to currentDF data frame.
4. I randomly generated values for prediction column in the range 0 and 1 ten times as we have ten rows for id, label, and weight, and added the prediction column to the currentDF data frame
5. Later I found the Error by using one condition i.e if the label is equal to a prediction it gives the result 0 otherwise it gives the result 1 and added the error column to the currentDF data frame
6. And then I found the product of weights and error and assigned that value to xy variable, and added the XY column to the currentDF data frame
7. Now I have to find the alpha value and the formula is 0.5 \* (log((1-sumofxy)/sumofxy))
8. By using the alpha values I have to find correct and incorrect values. The formulas for correct and incorrect are as follows Correct: exp(-alpha*1) Incorrect :exp(-alpha*-1)
9. Now finding the adjustment, if the label is equal to prediction the result will be correct otherwise the result will be incorrect, and added the adjustment column to the currentDF data frame
10. Later I found the adj\_weight which is the product of adjustment and weights and added the adj\_weight column to the currentDF data frame
11. Next, I found the new\_weight y using the formula adj\_weight/sum(adj\_weight) and assigned a new\_weight column to the currentDF data frame.
12. Now I am assigning the currentDF data frame to nextDF data frame and also assigning the values of new\_weights column of currentDF data frame values to nextDF data frame weights column and printing all the four iteration values of currentDF, here is the end of while loop
13. In question, it was asked to find the confusion matrix for the fourth iteration, so I am printing the currentDF out of a while loop which will give the fourth iteration values. I am making one table for label and prediction which will give the true positives and false positives. Add the true positives and true negatives and then divide by all the values to find the confusion matrix. Therefore I got the confusion matrix with accuracy 0.4 i.e 40%

### QUESTION(h)

Generate prediction probabilities obtained in your best model aboveand use R code to create and plot an ROC curve, note you can only used base R commands here no other packages are allowed. Comment on the ROC curve

#decision tree   
#The seed number is the starting point used in the generation of a sequence of random numbers, and the same results will be given if the same seed number is used.  
set.seed(214)#setting the seed with the last three digits of my student number  
#here I am making two samples by dividing the rows of Credit\_Training\_Data into 80% and 20% probability and storing in to variable id  
id<-sample(2,nrow(Credit\_Training\_Data),prob=c(0.8,0.2),replace=TRUE)  
  
#I am creating training data with the first sample(80%) i.e id==1 and storing that in to credit\_train  
Credit\_train=Credit\_Training\_Data[id==1,]  
#View(Credit\_train)  
nrow(Credit\_train)#here by using nrow we can see how many rows the credit\_train is taking

## [1] 633

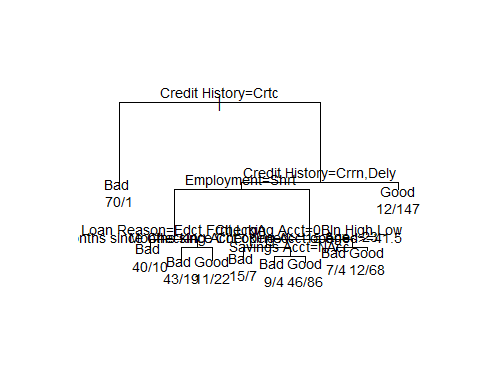
#after execution it show it is taking 633 rows for training the model  
  
#I am creating testing data with the second sample(20%) i.e id==2 and storing that in to credit\_test  
  
Credit\_test=Credit\_Training\_Data[id==2,]  
#View(Credit\_test)  
#here by using nrow we can see how many rows the credit\_train is taking  
nrow(Credit\_test)

## [1] 147

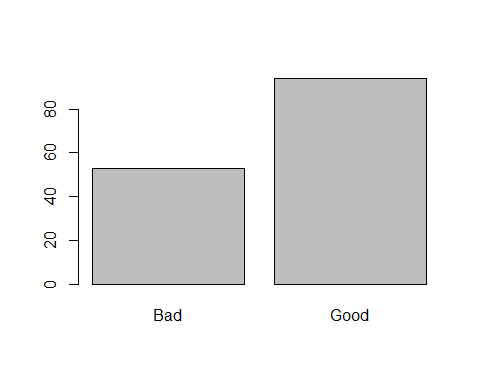
#after execution it show it is taking 147 rows for testing the model  
  
#now creating the model by using the credit standing column with the data credit\_train i.e training data(we will train the model by using this training data)  
#here i am using rpart package for creating the decision tree  
#the model is stored in the variable Credit\_model  
Credit\_model<-rpart(`Credit Standing`~.,data=Credit\_train)  
Credit\_model#viewing the model

## n= 633   
##   
## node), split, n, loss, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 633 265 Good (0.41864139 0.58135861)   
## 2) Credit History=Critical 71 1 Bad (0.98591549 0.01408451) \*  
## 3) Credit History=All Paid,Bank Paid,Current,Delay 562 195 Good (0.34697509 0.65302491)   
## 6) Credit History=Current,Delay 403 183 Good (0.45409429 0.54590571)   
## 12) Employment=Short 145 51 Bad (0.64827586 0.35172414)   
## 24) Loan Reason=Education,Furniture,Large Appliance 50 10 Bad (0.80000000 0.20000000) \*  
## 25) Loan Reason=Business,Car New,Car Used,Repairs,Small Appliance 95 41 Bad (0.56842105 0.43157895)   
## 50) Months since Checking Acct opened>=15.5 62 19 Bad (0.69354839 0.30645161) \*  
## 51) Months since Checking Acct opened< 15.5 33 11 Good (0.33333333 0.66666667) \*  
## 13) Employment=Long,Medium,Unemployed,Very Short 258 89 Good (0.34496124 0.65503876)   
## 26) Checking Acct=0Balance,High,Low 167 70 Good (0.41916168 0.58083832)   
## 52) Months since Checking Acct opened>=41.5 22 7 Bad (0.68181818 0.31818182) \*  
## 53) Months since Checking Acct opened< 41.5 145 55 Good (0.37931034 0.62068966)   
## 106) Savings Acct=No Acct 13 4 Bad (0.69230769 0.30769231) \*  
## 107) Savings Acct=High,Low,MedHigh,MedLow 132 46 Good (0.34848485 0.65151515) \*  
## 27) Checking Acct=No Acct 91 19 Good (0.20879121 0.79120879)   
## 54) Age< 23 11 4 Bad (0.63636364 0.36363636) \*  
## 55) Age>=23 80 12 Good (0.15000000 0.85000000) \*  
## 7) Credit History=All Paid,Bank Paid 159 12 Good (0.07547170 0.92452830) \*

plot(Credit\_model,margin=0.1)#plotting the Credit\_model,the plot will give the outline of the decision treei.e the graphical view of tree   
text(Credit\_model,use.n=TRUE,pretty=TRUE,cex=0.9)#the graphical view of the tree is filled with the text by using the text() function



#NOw predicting the model by using test data i.e by using credit\_test(we will always predict the values by using testing data)  
#Predicted values are stored in pred\_Credit variable  
pred\_Credit<-predict(Credit\_model,newdata=Credit\_test,type="class")  
plot(pred\_Credit)#I am plotting the predicted values to see the ratio of bad and good



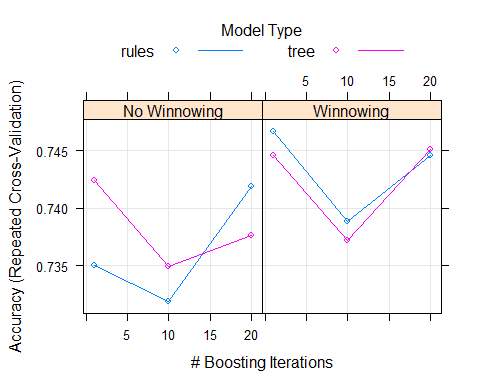
#Now creating the table for predicted values and actual test values i.e pred\_Credit and Credit\_test$`Credit Standing`  
table(pred\_Credit,Credit\_test$`Credit Standing`)

##   
## pred\_Credit Bad Good  
## Bad 34 19  
## Good 20 74

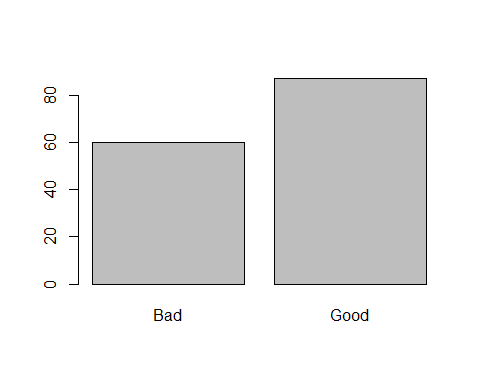
#now finding the confusion matrix by using confusionMatrix() function which is available in caret package.  
##generallt confusion matrix is the sum of true positive and true negatives divide by sum of all the values in the table.  
#finding confusion matrix for pred\_Credit and Credit\_test$`Credit Standing`  
confusionMatrix(table(pred\_Credit,Credit\_test$`Credit Standing`))

## Confusion Matrix and Statistics  
##   
##   
## pred\_Credit Bad Good  
## Bad 34 19  
## Good 20 74  
##   
## Accuracy : 0.7347   
## 95% CI : (0.6556, 0.804)  
## No Information Rate : 0.6327   
## P-Value [Acc > NIR] : 0.005698   
##   
## Kappa : 0.427   
##   
## Mcnemar's Test P-Value : 1.000000   
##   
## Sensitivity : 0.6296   
## Specificity : 0.7957   
## Pos Pred Value : 0.6415   
## Neg Pred Value : 0.7872   
## Prevalence : 0.3673   
## Detection Rate : 0.2313   
## Detection Prevalence : 0.3605   
## Balanced Accuracy : 0.7127   
##   
## 'Positive' Class : Bad   
##

#Among all the model boosting algorithm is giving highest accuracy  
# Boosting Algorithms  
  
#trainControl the computational quality that is not easy to notice but may be important to the train function  
#trainControl that allow us to perform variety of cross validation  
control <- trainControl(method="repeatedcv", number=10, repeats=3)  
  
set.seed(214)#setting the seed with the last three digits of my student number  
  
  
#now creating the model by using the credit standing column with the data credit\_train i.e training data(we will train the model by using this training data)  
#here i am using c50 package for boosting algorithm improving the decision tree model  
#the model is stored in the variable Boost\_model  
Boost\_model <- train(`Credit Standing`~., data=Credit\_train, method="C5.0", metric="Accuracy", trControl=control)  
plot(Boost\_model)#plotting the model



#NOw predicting the model by using test data i.e by using credit\_test(we will always predict the values by using testing data)  
#Predicting the values by using Boost\_model and Credit\_test data and storing in the variable predict\_boost  
predict\_boost=predict(Boost\_model,newdata=Credit\_test)  
plot(predict\_boost)#I am plotting the predicted values to see the ratio of bad and good



#Now creating the table for predicted values and actual test values i.e predict\_boost and Credit\_test$`Credit Standing`  
  
table(predict\_boost,Credit\_test$`Credit Standing`)

##   
## predict\_boost Bad Good  
## Bad 41 19  
## Good 13 74

#finding confusion matrix for predict\_boost and Credit\_test$`Credit Standing`  
  
confusionMatrix(table(predict\_boost,Credit\_test$`Credit Standing`))

## Confusion Matrix and Statistics  
##   
##   
## predict\_boost Bad Good  
## Bad 41 19  
## Good 13 74  
##   
## Accuracy : 0.7823   
## 95% CI : (0.7068, 0.8461)  
## No Information Rate : 0.6327   
## P-Value [Acc > NIR] : 6.843e-05   
##   
## Kappa : 0.5423   
##   
## Mcnemar's Test P-Value : 0.3768   
##   
## Sensitivity : 0.7593   
## Specificity : 0.7957   
## Pos Pred Value : 0.6833   
## Neg Pred Value : 0.8506   
## Prevalence : 0.3673   
## Detection Rate : 0.2789   
## Detection Prevalence : 0.4082   
## Balanced Accuracy : 0.7775   
##   
## 'Positive' Class : Bad   
##

#==============ROC curve  
  
#I am taking predict\_boost,if predict\_boost is equal to bad it takes 0(bad) else take 1(good)  
predict\_class\_roc <- ifelse(predict\_boost=="Bad",0,1)  
predict\_class\_roc#printing the predict\_class\_roc

## [1] 0 1 1 1 1 0 0 1 0 1 1 1 0 1 1 1 1 0 1 0 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0  
## [36] 1 0 1 0 1 1 1 1 1 1 0 0 1 0 0 0 0 1 1 0 1 1 0 1 1 1 0 1 0 1 0 0 0 0 1  
## [71] 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [106] 1 0 0 1 0 1 0 0 0 1 0 0 1 0 0 1 0 1 1 1 0 0 0 0 1 0 0 1 0 1 1 1 0 0 0  
## [141] 1 0 1 1 1 0 0

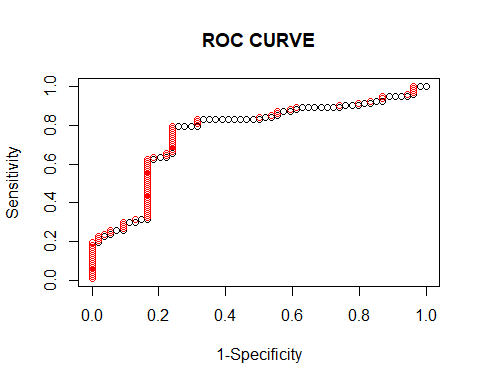
#I am taking Credit\_test$`Credit Standing`,if Credit\_test$`Credit Standing` is equal to bad it takes 0(bad) else take 1(good)  
predict\_class\_roc1 <- ifelse(Credit\_test$`Credit Standing`=="Bad",0,1)  
predict\_class\_roc1#printing the predict\_class\_roc

## [1] 0 1 1 1 1 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0  
## [36] 0 1 1 0 1 1 0 1 0 1 0 1 1 0 1 1 0 0 0 0 1 1 1 1 1 0 0 0 1 1 0 0 0 0 0  
## [71] 0 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1  
## [106] 0 1 0 0 0 1 0 1 0 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 1 0 1 1 1 1 1 1  
## [141] 1 1 1 1 1 0 0

#making the table for both predict\_class\_roc and predict\_class\_roc1  
table(predict\_class\_roc,predict\_class\_roc1)

## predict\_class\_roc1  
## predict\_class\_roc 0 1  
## 0 41 19  
## 1 13 74

#Creating the function ROC   
ROC <- function(predict\_class\_roc1,predict\_class\_roc)  
{  
 #now ordering the predict\_class\_roc1 and storing in predict\_class\_roc1  
 predict\_class\_roc1 <- predict\_class\_roc1[order(predict\_class\_roc,decreasing = TRUE)]  
 #now finding sensitivity(true positive) and specificity(false positive) and storing in the dataframe  
 data.frame(sensitivity=cumsum(predict\_class\_roc1)/sum(predict\_class\_roc1), specifity=cumsum(!predict\_class\_roc1)/sum(!predict\_class\_roc1),predict\_class\_roc1)  
}  
  
  
  
Roc\_CURVE <-ROC(predict\_class\_roc1,predict\_class\_roc)#Calling the Roc function by passing actual values and predicted values for plotting the graph and storing in Roc\_CURVE variable  
  
#plotting the Roc graph by taking the specificity on x-axis and sensitivity on y axis  
plot(Roc\_CURVE$specifity,Roc\_CURVE$sensitivity,col=1+Roc\_CURVE$predict\_class\_roc1,main="ROC CURVE",xlab="1-Specificity",ylab="Sensitivity")



In the question, it is given that generate prediction probabilities obtained in your best model above and plot a ROC curve.

I built the decision tree by using the rpart package and the accuracy is 73.47%

I used boosting and bagging algorithms to improve my decision tree model. Among those two boosting algorithms is giving good accuracy in the improvement of the tree.

So I am considering boosting algorithm and plotting Roc curve.

Here I built a decision tree and applying a boosting algorithm to it in order to explain clearly by showing both the models and after plotting the ROC curve

For plotting the ROC curve the steps are as followed:

1. I am taking predicted values of boosting algorithm and writing some condition i.e if predict value is equal to bad it takes 0(bad) else take 1(good). Basically I am converting good to 1 and bad to zero in order to calculate the true positives and false positives easily.
2. I am considering the actual values i.e the testing data values of credit standing column and repeating the same steps i.e if the actual value is equal to bad it takes 0(bad) else take 1(good).
3. Now writing one function and named as ROC which contains the calculations of sensitivity(true positives) and specificity(false positives). Firstly I will arrange the actual values in one order and then I am finding the sensitivity and specificity. Later I am storing those two values in a data frame.
4. After that, I am calling ROC function bypassing actual values and predicted values and storing it in one variable called Roc\_CURVE
5. Finally, I am plotting the ROC plot by taking the specificity on x\_axis and sensitivity on y\_axis In the graph, the actual values are in red color and predicted values are in black color

From the graph we can say that almost the predicted and actual values are the same with a small number of values are different. By looking at the graph we can say that it has the best performance as it is not close to the baseline.

References:

1.https://blog.revolutionanalytics.com/2016/08/roc-curves-in-two-lines-of-code.ht