Data Science and Analytics Assignment

**Reading XLSX data using XLConnect Package**

credit\_train\_df=readWorksheetFromFile("Credit\_Risk6\_final.xlsx", sheet = "Training\_Data", header = TRUE)  
head(credit\_train\_df,n = 3)

## ID Checking.Acct Credit.History Loan.Reason Savings.Acct Employment  
## 1 1 No Acct All Paid Car New Low Medium  
## 2 2 0Balance Current Car New Low Short  
## 3 3 0Balance Current Car New No Acct Long  
## Personal.Status Housing Job.Type Foreign.National  
## 1 Single Own Management No  
## 2 Divorced Own Skilled No  
## 3 Divorced Own Skilled No  
## Months.since.Checking.Acct.opened. Residence.Time..In.current.district.  
## 1 7 3  
## 2 16 2  
## 3 25 2  
## Age Credit.Standing  
## 1 44 Good  
## 2 28 Bad  
## 3 28 Bad

credit\_test\_df=readWorksheetFromFile("Credit\_Risk6\_final.xlsx", sheet = "Scoring\_Data", header = TRUE)  
head(credit\_test\_df,n=3)

## ID Checking.Acct Credit.History Loan.Reason Savings.Acct Employment  
## 1 781 No Acct All Paid Car New MedHigh Short  
## 2 782 Low Current Small Appliance Low Medium  
## 3 783 No Acct Current Small Appliance Low Very Short  
## Personal.Status Housing Job.Type Foreign.National  
## 1 Single Rent Unskilled No  
## 2 Single Rent Skilled No  
## 3 Divorced Own Skilled No  
## Months.since.Checking.Acct.opened. Residence.Time Age  
## 1 11 2 39  
## 2 37 4 23  
## 3 13 2 28

## Preprocessing of Data

**1) Checking NA values**: **Employment has 33 NA, Personal Status has 6 NA and Housing has 5 NA. We will impute this NA values**

apply(credit\_train\_df,2,function(x) sum(is.na(x)))

## ID Checking.Acct   
## 0 0   
## Credit.History Loan.Reason   
## 0 0   
## Savings.Acct Employment   
## 0 33   
## Personal.Status Housing   
## 6 5   
## Job.Type Foreign.National   
## 0 0   
## Months.since.Checking.Acct.opened. Residence.Time..In.current.district.   
## 0 0   
## Age Credit.Standing   
## 0 0

**2) Employment Type *Short* has 242 occurences so we will impute it**

sort(table(credit\_train\_df$Employment),decreasing = T)

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

*we will impute the NA values with Short type for Employment column*

credit\_train\_df$Employment=replace\_na(credit\_train\_df$Employment,'Short')

**3) Personal Status has Single type with 431 rows**

sort(table(credit\_train\_df$Personal.Status),decreasing = T)

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

*we will impute the NA values with Single type for Personal Status column*

credit\_train\_df$Personal.Status=replace\_na(credit\_train\_df$Personal.Status,'Single')

**4) Housing has Single with 524 rows**

sort(table(credit\_train\_df$Housing),decreasing = T)

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

*we will impute the NA values with Own type for Housing column*

credit\_train\_df$Housing=replace\_na(credit\_train\_df$Housing,'Own')

*Now we can see that we have no NA values after imputing with maximum count*

apply(credit\_train\_df,2,function(x) sum(is.na(x)))

## ID Checking.Acct   
## 0 0   
## Credit.History Loan.Reason   
## 0 0   
## Savings.Acct Employment   
## 0 0   
## Personal.Status Housing   
## 0 0   
## Job.Type Foreign.National   
## 0 0   
## Months.since.Checking.Acct.opened. Residence.Time..In.current.district.   
## 0 0   
## Age Credit.Standing   
## 0 0

**Making the column name common for Both the dataset**

colnames(credit\_train\_df)[12]=colnames(credit\_test\_df[12])

**converting all the character class to convert it into factors**

credit\_train\_df[credit\_factor\_train] <- lapply(credit\_train\_df[credit\_factor\_train], factor)  
credit\_test\_df[credit\_factor\_test] <- lapply(credit\_test\_df[credit\_factor\_test], factor)

## Question1: Trivariate analysis

**Finding the cross tabulation between Saving Account, Credit History and Credit Standing**

xtabs(~Checking.Acct+Credit.History+Credit.Standing,data=credit\_train\_df)

## , , Credit.Standing = Bad  
##   
## Credit.History  
## Checking.Acct All Paid Bank Paid Critical Current Delay  
## 0Balance 0 5 38 75 9  
## High 0 1 8 8 3  
## Low 1 2 28 59 5  
## No Acct 4 2 17 45 9  
##   
## , , Credit.Standing = Good  
##   
## Credit.History  
## Checking.Acct All Paid Bank Paid Critical Current Delay  
## 0Balance 28 25 0 82 1  
## High 6 4 0 14 1  
## Low 17 18 0 56 15  
## No Acct 82 6 1 84 21

xtabs(~Savings.Acct+Credit.History+Credit.Standing,data=credit\_train\_df)

## , , Credit.Standing = Bad  
##   
## Credit.History  
## Savings.Acct All Paid Bank Paid Critical Current Delay  
## High 1 3 0 5 0  
## Low 3 0 69 118 26  
## MedHigh 0 2 8 10 0  
## MedLow 0 0 6 19 0  
## No Acct 1 5 8 35 0  
##   
## , , Credit.Standing = Good  
##   
## Credit.History  
## Savings.Acct All Paid Bank Paid Critical Current Delay  
## High 9 1 0 8 2  
## Low 77 37 1 155 14  
## MedHigh 4 2 0 19 4  
## MedLow 8 12 0 25 10  
## No Acct 35 1 0 29 8

*For Credit standing Good, we have more most people with paid loans compared to Bad type.*

## Question2: Decision Tree

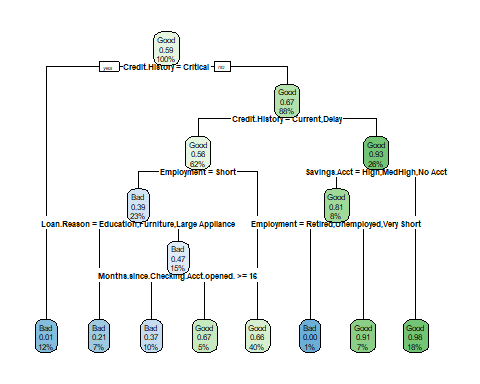
**Step1) Building the Decision Tree on Train Dataset**

credit\_RDT\_model=rpart(Credit.Standing~.,data = credit\_train\_df[,-1])  
credit\_RDT\_model

## n= 780   
##   
## node), split, n, loss, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 780 319 Good (0.40897436 0.59102564)   
## 2) Credit.History=Critical 92 1 Bad (0.98913043 0.01086957) \*  
## 3) Credit.History=All Paid,Bank Paid,Current,Delay 688 228 Good (0.33139535 0.66860465)   
## 6) Credit.History=Current,Delay 487 213 Good (0.43737166 0.56262834)   
## 12) Employment=Short 177 69 Bad (0.61016949 0.38983051)   
## 24) Loan.Reason=Education,Furniture,Large Appliance 57 12 Bad (0.78947368 0.21052632) \*  
## 25) Loan.Reason=Business,Car New,Car Used,Other,Repairs,Small Appliance 120 57 Bad (0.52500000 0.47500000)   
## 50) Months.since.Checking.Acct.opened.>=15.5 78 29 Bad (0.62820513 0.37179487) \*  
## 51) Months.since.Checking.Acct.opened.< 15.5 42 14 Good (0.33333333 0.66666667) \*  
## 13) Employment=Long,Medium,Unemployed,Very Short 310 105 Good (0.33870968 0.66129032) \*  
## 7) Credit.History=All Paid,Bank Paid 201 15 Good (0.07462687 0.92537313)   
## 14) Savings.Acct=High,MedHigh,No Acct 64 12 Good (0.18750000 0.81250000)   
## 28) Employment=Retired,Unemployed,Very Short 7 0 Bad (1.00000000 0.00000000) \*  
## 29) Employment=Long,Medium,Short 57 5 Good (0.08771930 0.91228070) \*  
## 15) Savings.Acct=Low,MedLow 137 3 Good (0.02189781 0.97810219) \*

**Step2) Plotting the Decision Tree**

rpart.plot(credit\_RDT\_model,cex=0.5,roundint = FALSE)



# text(credit\_RDT\_model,pretty=0, cex=0.5)

**Step3) Predicting the model on Train dataset**

credit\_RDT\_predict=predict(credit\_RDT\_model,newdata = credit\_train\_df[-1],type = c('class'))

**Step4) Finding the model performance using Accuracy of the model**

confusion\_Mat=table(credit\_train\_df$Credit.Standing,credit\_RDT\_predict)  
Accuracy\_DT <- sum(diag(confusion\_Mat))/sum(confusion\_Mat)  
Accuracy\_DT

## [1] 0.7833333

**Step4) Predicting the model on Scoring dataset: We can take the prediction value as 78%**

credit\_RDT\_pred\_scoring=predict(credit\_RDT\_model,credit\_test\_df,type = "class")  
credit\_RDT\_pred\_scoring

## 1 2 3 4 5 6 7 8 9 10 11 12 13   
## Good Good Good Good Good Good Good Bad Good Good Bad Bad Bad   
## Levels: Bad Good

## Question3: Explaining the Decision Tree

**We will first predict the data on the scoring dataset.**

credit\_RDT\_pred\_scoring=predict(credit\_RDT\_model,credit\_test\_df,type = "class")  
credit\_RDT\_pred\_scoring

## 1 2 3 4 5 6 7 8 9 10 11 12 13   
## Good Good Good Good Good Good Good Bad Good Good Bad Bad Bad   
## Levels: Bad Good

## Explaining the decision tree

**Basics of Decision Tree:** *Decision Tree algorithm works in a form of leaf/root node concepts.Each node makes decision* *of YES or NO.In Decision tree, a concept of Enthropy/Gini Index/Information gain is used to find the column to* *be selected for making decision.*

**For instance:** **ROW1**: *Let us consider a column from Scoring dataset with values as listed below.* *ID: 781 and Credit History: All Paid* *Credit History has minimum Entropy out of all columns, so decision tree is the column got selected. When Credit History* *is of type ‘All Paid’ OR ‘Current’, decision goes to left and if the Credit History is Critical, decision goes to right.* *So if we consider first column, Credit History is All Paid so the final decision is BAD as can be read from the* *tree.*

**ROW2:** *Let us consider second row with values as below* *ID: 781, Credit History: Critical, Employment: Medium, Loan Reason: Car Used In the second level, Credit History again* *got selected with minimum entropy. So as our column, contains Credit History as Critical, tree will be moved to the* *Left. Here, Employment got selected with minimum entropy, so if it is Medium tree takes it to the next node to the left.* *Now, it is been judged based on the value of Loan Reason as the least entropy. Our data has Loan Reason as Car* *used it will directly take the final decision to the left with Bad.*

**ROW3:** *If we consider row with ID: 782 Credit History as Current, it will take the final decision of BAD directly by* *considering the left path.*

**ROW4:** *If we consider row with ID: 792, Credit History as Critical will take the tree to the left.* *Employment as Short will take the tree to the right which directly makes the final decision of GOOD.*

**ROW5:** *If we consider row with ID: 783, Credit History as Current will take the tree to the left.* *Decision is made with result as BAD*

## Question4: Random Forest & Boosting Technique

**Step1) Building the Random Forest model with train data set**

credit\_RF\_model=randomForest(Credit.Standing~.-ID,data=credit\_train\_df,ntree=100,mtry=5)  
credit\_RF\_model

##   
## Call:  
## randomForest(formula = Credit.Standing ~ . - ID, data = credit\_train\_df, ntree = 100, mtry = 5)   
## Type of random forest: classification  
## Number of trees: 100  
## No. of variables tried at each split: 5  
##   
## OOB estimate of error rate: 26.54%  
## Confusion matrix:  
## Bad Good class.error  
## Bad 216 103 0.3228840  
## Good 104 357 0.2255965

**Step2) Predicting on train dataset: Model accuracy is 93%**

credit\_RF\_Pred\_train=predict(credit\_RF\_model,newdata=credit\_train\_df,type="response")  
RF\_confmat=table(credit\_train\_df$Credit.Standing,credit\_RF\_Pred\_train)  
RF\_Accuracy\_Train <- sum(diag(RF\_confmat))/sum(RF\_confmat)  
RF\_Accuracy\_Train

## [1] 0.9384615

**Step3) Predicting on Scoring datasets**

credit\_RF\_Pred\_scoring=predict(credit\_RF\_model,newdata=credit\_test\_df,type="response")  
credit\_RF\_Pred\_scoring

## 2 3 4 5 6 7 8 9 10 11 12 13 14   
## Good Good Good Good Good Good Good Good Good Good Bad Bad Bad   
## Levels: Bad Good

**Step4) Building the Boosting model with train dataset**

credit\_train\_df=credit\_train\_df %>% mutate(Credit.Standing\_10=as.numeric(Credit.Standing=="Good"))  
credit\_GBM\_model=gbm(Credit.Standing\_10~.,data=credit\_train\_df[c(-1,-14)],distribution = "multinomial",  
 n.trees = 100,interaction.depth = 3)

**Step5) Predicting on Train dataset**

credit\_GBM\_Pred\_train=predict.gbm(object = credit\_GBM\_model,  
 newdata = credit\_train\_df[c(-1,-14,-15)],  
 n.trees = 100,  
 type = "response")  
credit\_GBM\_Pred\_train\_val=ifelse(credit\_GBM\_Pred\_train[781:1560]<0.5,0,1)  
boost\_confmat=table(credit\_train\_df$Credit.Standing,credit\_GBM\_Pred\_train\_val)  
Boost\_Accuracy\_Train <- sum(diag(boost\_confmat))/sum(boost\_confmat)  
Boost\_Accuracy\_Train

## [1] 0.8512821

**Step6) Predicting on Scoring Datasets**

credit\_GBM\_Pred\_train=predict.gbm(object = credit\_GBM\_model,  
 newdata = credit\_train\_df[c(-1,-14,-15)],  
 n.trees = 100,  
 type = "response")  
pred = predict.gbm(object = credit\_GBM\_model,  
 newdata = credit\_test\_df,  
 n.trees = 200,  
 type = "response")

## Warning in predict.gbm(object = credit\_GBM\_model, newdata =  
## credit\_test\_df, : Number of trees not specified or exceeded number fit so  
## far. Using 100.

credit\_GBM\_Pred\_scoring=ifelse(pred[14:26]<0.5,'Bad','Good')  
credit\_GBM\_Pred\_scoring

## [1] "Good" "Bad" "Good" "Good" "Good" "Good" "Good" "Good" "Good" "Good"  
## [11] "Bad" "Bad" "Bad"

## Question5: Pattern in the dataset

*we will take Decision Tree to predict the pattern in the dataset.* *We have taken predicted values of Decision true and check where there is a consecutive incorrect results in the* *result set. We could see from the below result that the predicted values of Id’s 299 to 336 has incorrect results*. *Her getting suspicious about the incorrect pattern appears to be a valid reason.*

credit\_train\_df=cbind(credit\_train\_df,Predicted=credit\_RDT\_predict)  
credit\_train\_df[which(credit\_train\_df$Credit.Standing!=credit\_RDT\_predict & credit\_train\_df$ID %in% 304:326),c("ID","Credit.Standing","Predicted")]

## ID Credit.Standing Predicted  
## 304 304 Bad Good  
## 305 305 Bad Good  
## 306 306 Bad Good  
## 307 307 Bad Good  
## 308 308 Bad Good  
## 309 309 Good Bad  
## 310 310 Good Bad  
## 311 311 Good Bad  
## 312 312 Bad Good  
## 314 314 Bad Good  
## 315 315 Bad Good  
## 316 316 Bad Good  
## 317 317 Bad Good  
## 318 318 Bad Good  
## 319 319 Bad Good  
## 320 320 Bad Good  
## 322 322 Bad Good  
## 323 323 Bad Good  
## 324 324 Bad Good  
## 325 325 Bad Good  
## 326 326 Bad Good

## Question6: Infogain algorithm

**Step1) Create the Entropy for dependent variable ‘Credit.Standing’ so to subtract it to the Independent column to get Information gain**

DV\_Prop <- prop.table(table(credit\_train\_df$Credit.Standing))  
DV\_Entropy=sum(-DV\_Prop\*log2(DV\_Prop))  
DV\_Entropy

## [1] 0.9759588

**Step2) Find the proportion of independent variables wrt Credit.Standing**

table\_func <- function(x) {prop.table(table(credit\_train\_df[,x],credit\_train\_df[,14]) + 1e-6, margin = 1)}   
print(table\_func(4))

##   
## Bad Good  
## Business 3.658537e-01 6.341463e-01  
## Car New 4.450262e-01 5.549738e-01  
## Car Used 3.466667e-01 6.533333e-01  
## Education 5.333333e-01 4.666667e-01  
## Furniture 4.812500e-01 5.187500e-01  
## Large Appliance 3.750000e-01 6.250000e-01  
## Other 3.636364e-01 6.363636e-01  
## Repairs 2.000000e-01 8.000000e-01  
## Retraining 4.999995e-07 9.999995e-01  
## Small Appliance 3.548387e-01 6.451613e-01

**Step3) Apply the Entropy formulae i.e. -p*log2(p)-q*log2(q) for respect columns where p=probability of occurence** **of an event and q=1-p**

-table\_func(4)\*log2(table\_func(4))

##   
## Bad Good  
## Business 5.307298e-01 4.167054e-01  
## Car New 5.198074e-01 4.714549e-01  
## Car Used 5.298380e-01 4.012178e-01  
## Education 4.836750e-01 5.131166e-01  
## Furniture 5.077869e-01 4.911985e-01  
## Large Appliance 5.306391e-01 4.237950e-01  
## Other 5.307024e-01 4.149579e-01  
## Repairs 4.643856e-01 2.575425e-01  
## Retraining 1.046577e-05 7.213466e-07  
## Small Appliance 5.304004e-01 4.079150e-01

**Step4) Rowsum all the categories of a column to find the individual category Entropy values**

rowSums(-table\_func(10)\*log2(table\_func(10)))

## No Yes   
## 0.9809722 0.9733402

**Step5) Find the weight of that column and multiply it with individual category entropy to find complete entropy Of that column**

prop.table(table(credit\_train\_df$Foreign.National))

##   
## No Yes   
## 0.324359 0.675641

**Step6) Below code will find the entropy for a particular column by multiply each categories to its weight and summing** **all the categories to find overall column entropy**

sum(prop.table(table(credit\_train\_df$Foreign.National))\*rowSums(-table\_func(10)\*log2(table\_func(10))))

## [1] 0.9758157

**Step7) Summing up above code to find the Information Gain using Entropy i.e. INFO GAIN=Entropy of the Dependent** **variable - Entropy of Independent of all the columns by a single function. We have calculated the Information gain by** **subtracting Entropy of a column with Entropy of Credit Standing.**

Column\_Entropy\_New <- function(x) {   
 table\_func <- prop.table(table(credit\_train\_df[,x],credit\_train\_df[,14]) + 1e-6, margin = 1)  
 DV\_Entropy-sum(prop.table(table(credit\_train\_df[,x]))\*rowSums(-table\_func\*log2(table\_func)))}

**Step8) We have found the maximum of all the Entropy to decide which column to be selected as the column of split in a** **Decision Tree**

Information\_Gain=sapply(credit\_factor\_test,Column\_Entropy\_New)  
Information\_Gain[which.max(Information\_Gain)]

## Credit.History   
## 0.2580026

## Question7: Adaboost Algorithm

*Boosting Algorithm is based on the technique sequential model process where the error of one model acts as an dependent variable to the next model.In case of Adaboost, we will initially assign a default weight to each row based on* *1/number of rows. We will find the errors based on predicted and actual values. We will find the value of the amount* \_of say i.e. Alpha with the formulae alpha=0.5\*ln((1-error\_weight)/error\_weight). We will assign this alpha value\_ *and find adjustment using the formulae as: if error is 1, e^-alpha and if the error is 0, e^-alpha.* \_We will find the adjusted weight as original weight\*adjustment. We will normalise the result as New\_Weights\_. *This new weights acts as an Weight for next stump. This process repeats for 4 iteration in our example.* *In the end, the final decision is been made based on the weight assign to each Stump*.

adaboost\_list <- vector('list', 4)  
for(iterations in seq(1,4)){  
 set.seed(iterations)  
 predicted=sample(c(0,1), replace=TRUE, size=10)  
 name=paste("adaboost",iterations,"df",sep = "\_")  
   
 if(iterations==1){  
 adaboost\_list[[name]]=data.frame(Id=1:10, Label=c(0,1,1,0,1,1,0,1,0,0),Weights=rep(0.1,10),  
 P1=c(0,1,0,0,0,0,1,1,0,0),Prediction=predicted)  
 }  
   
 else{  
 adaboost\_list[[name]]=data.frame(Id=1:10, Label=c(0,1,1,0,1,1,0,1,0,0),Weights=adaboost\_list[[previous\_name]]$New\_Weights,  
 P1=c(0,1,0,0,0,0,1,1,0,0),Prediction=predicted)  
 }  
 adaboost\_list[[name]]=cbind(adaboost\_list[[name]],  
 Error=ifelse(adaboost\_list[[name]]$Prediction==adaboost\_list[[name]]$Label,0,1))  
 adaboost\_list[[name]]=cbind(adaboost\_list[[name]],Er\_Wgt=adaboost\_list[[name]]$Weights\*adaboost\_list[[name]]$Error)  
   
 alpha\_1=0.5\*log((1-sum(adaboost\_list[[name]]$Er\_Wgt))/sum(adaboost\_list[[name]]$Er\_Wgt))  
 print(alpha\_1)  
 adaboost\_list[[name]]=cbind(adaboost\_list[[name]],Adjustment=ifelse(adaboost\_list[[name]]$Error==1,  
 exp(alpha\_1),exp(-alpha\_1)))  
 adaboost\_list[[name]]=cbind(adaboost\_list[[name]],Adj\_Wgt=adaboost\_list[[name]]$Adjustment\*adaboost\_list[[name]]$Weights)  
 adaboost\_list[[name]]=cbind(adaboost\_list[[name]],New\_Weights=adaboost\_list[[name]]$Adj\_Wgt/sum(adaboost\_list[[name]]$Adj\_Wgt))  
 previous\_name=name  
}

## [1] 0  
## [1] 0.2027326  
## [1] 0.3465736  
## [1] 0.06258157

adaboost\_list

## [[1]]  
## NULL  
##   
## [[2]]  
## NULL  
##   
## [[3]]  
## NULL  
##   
## [[4]]  
## NULL  
##   
## $adaboost\_1\_df  
## Id Label Weights P1 Prediction Error Er\_Wgt Adjustment Adj\_Wgt  
## 1 1 0 0.1 0 0 0 0.0 1 0.1  
## 2 2 1 0.1 1 1 0 0.0 1 0.1  
## 3 3 1 0.1 0 0 1 0.1 1 0.1  
## 4 4 0 0.1 0 0 0 0.0 1 0.1  
## 5 5 1 0.1 0 1 0 0.0 1 0.1  
## 6 6 1 0.1 0 0 1 0.1 1 0.1  
## 7 7 0 0.1 1 0 0 0.0 1 0.1  
## 8 8 1 0.1 1 0 1 0.1 1 0.1  
## 9 9 0 0.1 0 1 1 0.1 1 0.1  
## 10 10 0 0.1 0 1 1 0.1 1 0.1  
## New\_Weights  
## 1 0.1  
## 2 0.1  
## 3 0.1  
## 4 0.1  
## 5 0.1  
## 6 0.1  
## 7 0.1  
## 8 0.1  
## 9 0.1  
## 10 0.1  
##   
## $adaboost\_2\_df  
## Id Label Weights P1 Prediction Error Er\_Wgt Adjustment Adj\_Wgt  
## 1 1 0 0.1 0 0 0 0.0 0.8164966 0.08164966  
## 2 2 1 0.1 1 0 1 0.1 1.2247449 0.12247449  
## 3 3 1 0.1 0 1 0 0.0 0.8164966 0.08164966  
## 4 4 0 0.1 0 1 1 0.1 1.2247449 0.12247449  
## 5 5 1 0.1 0 1 0 0.0 0.8164966 0.08164966  
## 6 6 1 0.1 0 1 0 0.0 0.8164966 0.08164966  
## 7 7 0 0.1 1 0 0 0.0 0.8164966 0.08164966  
## 8 8 1 0.1 1 0 1 0.1 1.2247449 0.12247449  
## 9 9 0 0.1 0 0 0 0.0 0.8164966 0.08164966  
## 10 10 0 0.1 0 1 1 0.1 1.2247449 0.12247449  
## New\_Weights  
## 1 0.08333333  
## 2 0.12500000  
## 3 0.08333333  
## 4 0.12500000  
## 5 0.08333333  
## 6 0.08333333  
## 7 0.08333333  
## 8 0.12500000  
## 9 0.08333333  
## 10 0.12500000  
##   
## $adaboost\_3\_df  
## Id Label Weights P1 Prediction Error Er\_Wgt Adjustment  
## 1 1 0 0.08333333 0 0 0 0.00000000 0.7071068  
## 2 2 1 0.12500000 1 1 0 0.00000000 0.7071068  
## 3 3 1 0.08333333 0 1 0 0.00000000 0.7071068  
## 4 4 0 0.12500000 0 0 0 0.00000000 0.7071068  
## 5 5 1 0.08333333 0 1 0 0.00000000 0.7071068  
## 6 6 1 0.08333333 0 1 0 0.00000000 0.7071068  
## 7 7 0 0.08333333 1 1 1 0.08333333 1.4142136  
## 8 8 1 0.12500000 1 0 1 0.12500000 1.4142136  
## 9 9 0 0.08333333 0 0 0 0.00000000 0.7071068  
## 10 10 0 0.12500000 0 1 1 0.12500000 1.4142136  
## Adj\_Wgt New\_Weights  
## 1 0.05892557 0.06250  
## 2 0.08838835 0.09375  
## 3 0.05892557 0.06250  
## 4 0.08838835 0.09375  
## 5 0.05892557 0.06250  
## 6 0.05892557 0.06250  
## 7 0.11785113 0.12500  
## 8 0.17677670 0.18750  
## 9 0.05892557 0.06250  
## 10 0.17677670 0.18750  
##   
## $adaboost\_4\_df  
## Id Label Weights P1 Prediction Error Er\_Wgt Adjustment Adj\_Wgt  
## 1 1 0 0.06250 0 1 1 0.06250 1.0645813 0.06653633  
## 2 2 1 0.09375 1 0 1 0.09375 1.0645813 0.09980450  
## 3 3 1 0.06250 0 0 1 0.06250 1.0645813 0.06653633  
## 4 4 0 0.09375 0 0 0 0.00000 0.9393364 0.08806279  
## 5 5 1 0.06250 0 0 1 0.06250 1.0645813 0.06653633  
## 6 6 1 0.06250 0 1 0 0.00000 0.9393364 0.05870853  
## 7 7 0 0.12500 1 0 0 0.00000 0.9393364 0.11741705  
## 8 8 1 0.18750 1 1 0 0.00000 0.9393364 0.17612558  
## 9 9 0 0.06250 0 0 0 0.00000 0.9393364 0.05870853  
## 10 10 0 0.18750 0 1 1 0.18750 1.0645813 0.19960899  
## New\_Weights  
## 1 0.06666667  
## 2 0.10000000  
## 3 0.06666667  
## 4 0.08823529  
## 5 0.06666667  
## 6 0.05882353  
## 7 0.11764706  
## 8 0.17647059  
## 9 0.05882353  
## 10 0.20000000

## Question8: ROC Curve

**Step1) Considering the Boosting model to find the probabilities prediction**

gbm\_pred\_1=credit\_GBM\_Pred\_train[781:1560]

**Step2) Defining the cutoff dataframe**

cutoff\_data=data.frame(cutoff=0,TP=0,FP=0,FN=0,TN=0)

**Step3) Creating cutoffs between 0 to 1 100 values**

cutoffs=round(seq(0,1,length=100),3)

**Step4) Looping to set the TP,FP, TN, FN**

for (cutoff in cutoffs){  
 predicted=as.numeric(gbm\_pred\_1>cutoff)  
   
 TP=sum(predicted==1 & credit\_train\_df$Credit.Standing\_10==1)  
 FP=sum(predicted==1 & credit\_train\_df$Credit.Standing\_10==0)  
 FN=sum(predicted==0 & credit\_train\_df$Credit.Standing\_10==1)  
 TN=sum(predicted==0 & credit\_train\_df$Credit.Standing\_10==0)  
 cutoff\_data=rbind(cutoff\_data,c(cutoff,TP,FP,FN,TN))  
}

**Step5) Setting the different matrix e.g. Specificity, Sensitivity, KS, Accuracy based on the cutoff dataframe**

cutoff\_data=cutoff\_data %>%  
 mutate(Sn=TP/P, Sp=TN/N,  
 dist=sqrt((1-Sn)\*\*2+(1-Sp)\*\*2),  
 P=FN+TP,N=TN+FP) %>%  
 mutate(KS=abs((TP/P)-(FP/N))) %>%  
 mutate(Accuracy=(TP+TN)/(P+N)) %>%  
 mutate(Lift=(TP/P)/((TP+FP)/(P+N))) %>%  
 mutate(M=(8\*FN+2\*FP)/(P+N)) %>%  
 select(-P,-N)

**Step6) Creating ROC dataframe with True positive rate as Sensitivity and False Positive Rate as 1-Specificity**

roc\_data=cutoff\_data %>%   
 select(cutoff,Sn,Sp) %>%   
 mutate(TPR=Sn,FPR=1-Sp) %>%   
 select(cutoff,TPR,FPR)

**Step7) # Plotting the ROC AUC Curve**

plot(roc\_data$FPR,roc\_data$TPR,main="ROC AUC Curve",xlab="True Positive Rate",ylab="False Positive Rate")  
lines(roc\_data$FPR,roc\_data$TPR)

