**Prediction of Credit Worthiness of loan for future potential customers**

**Part 2**

# knitr::opts\_chunk$set(echo = TRUE)  
##########################info gain###############################

tablefun <- function(x) ##created function named as tablefun  
{  
 p=prop.table(table(data2[,x],data2[,13])+1e-6,margin = 1)##to find the probability of each element of a column with respect to target variable  
   
 log\_p=p\*log2(p) #formula to calculate entropy  
 x1=rowSums(log\_p) # row wise summition  
 p1=prop.table(table(data2[,x]))# proportion in each of the rows  
 e=-sum(p1\*x1)# to calculate entropy need to multiply probability of each element with rowsums(log p)then sum of all element  
 ##print(e)  
 return(e)  
}  
ncol(data2)

## [1] 15

tablefun(3)

## [1] 0.7273546

##tablefun(6)  
infoGain=function(x)  
{  
 p=prop.table(table(data2[,13]))  
 e=p\*log2(p)  
 e\_creditstanding=sum(-e)  
 gain=e\_creditstanding-tablefun(x)# info gain formula.  
 print(gain)  
}  
infoGain(3)

## [1] 0.2511182

##create loop for the categorical data  
r <- 2:10  
  
r3 <- NULL  
for (x in r) {   
 r2 <- infoGain(x)  
r3 <- c(r3,r2)  
print(r3)  
}

## [1] 0.02901242  
## [1] 0.02901242  
## [1] 0.2511182  
## [1] 0.02901242 0.25111824  
## [1] 0.01784408  
## [1] 0.02901242 0.25111824 0.01784408  
## [1] 0.005996613  
## [1] 0.029012415 0.251118242 0.017844082 0.005996613  
## [1] 0.05430079  
## [1] 0.029012415 0.251118242 0.017844082 0.005996613 0.054300786  
## [1] 0.006197113  
## [1] 0.029012415 0.251118242 0.017844082 0.005996613 0.054300786 0.006197113  
## [1] 0.006227663  
## [1] 0.029012415 0.251118242 0.017844082 0.005996613 0.054300786 0.006197113  
## [7] 0.006227663  
## [1] 0.01142379  
## [1] 0.029012415 0.251118242 0.017844082 0.005996613 0.054300786 0.006197113  
## [7] 0.006227663 0.011423791  
## [1] 0.0004366205  
## [1] 0.0290124155 0.2511182417 0.0178440816 0.0059966130 0.0543007864  
## [6] 0.0061971135 0.0062276629 0.0114237915 0.0004366205

Entropy : Entropy measures the level of impurity .

Entropy=

Pi is the probability of class

Information Gain tells how important a given attribute of the feature vector.

InfoGain= Entropy of parent node- Entropy of child node.

Here from the highest infoGain is 0.2511 of Credit History says that it’s the most important attributes in the model.

##############################AdabagBoosting##################  
set.seed(158)  
adadata<-data.frame(Label=c(0,1,1,0,1,1,0,1,0,0),Weight=c(0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1))  
##for start taking 10 label   
#adadata<-data.frame(adadata,predi=sample(0:1,10,replace = T))  
View(adadata)  
for (i in seq(1,4)) {##for 4 times repetation  
   
 for(j in seq(1,10))  
 {  
 adadata$predi[j]<-sample(0:1,1,replace = T)#creating every time prediction value  
 if(adadata$Label[j]==adadata$predi[j])#checking if lebel equal to prediction and assign error as 0 if not error as 1  
 {  
 adadata$Error[j]<-0  
   
 }else{  
 adadata$Error[j]<-1  
 }  
 adadata$value[j]<-adadata$Error[j]\*adadata$Weight[j]#creating value as product of error and weight  
 }  
 sum(adadata$value)  
 alpha1=0.5\*log((1-sum(adadata$value))/sum(adadata$value))#alpha formula  
 incorrect=exp(-alpha1\*-1)#incorrect formula  
 correct=exp(-alpha1\*1)#correct formula  
 for (j in seq(1,10)) {  
   
 if(adadata$Label[j]==adadata$predi[j]) ##if label equal to prediction then assign adjusment equal to correct if not then assign incorrect.  
 {  
 adadata$adjustment[j]<-correct  
   
 }else{  
 adadata$adjustment[j]<-incorrect  
 }  
   
 adadata$product[j]<-adadata$Weight[j]\*adadata$adjustment[j]#product of weight and adjustment  
   
   
 }  
 sum(adadata$product)  
 for (j in seq(1,10)) {  
 adadata$NewWeight[j]<-adadata$product[j]/(sum(adadata$product))#new weight is product devide by sum of product  
   
 }  
 print(adadata)#printing data frame  
 adadata$Weight<-adadata$NewWeight #asssigning new weight to the weight  
   
}

## Label Weight predi Error value adjustment product NewWeight  
## 1 0 0.1 0 0 0.0 0.8164966 0.08164966 0.08333333  
## 2 1 0.1 0 1 0.1 1.2247449 0.12247449 0.12500000  
## 3 1 0.1 1 0 0.0 0.8164966 0.08164966 0.08333333  
## 4 0 0.1 0 0 0.0 0.8164966 0.08164966 0.08333333  
## 5 1 0.1 0 1 0.1 1.2247449 0.12247449 0.12500000  
## 6 1 0.1 1 0 0.0 0.8164966 0.08164966 0.08333333  
## 7 0 0.1 0 0 0.0 0.8164966 0.08164966 0.08333333  
## 8 1 0.1 1 0 0.0 0.8164966 0.08164966 0.08333333  
## 9 0 0.1 1 1 0.1 1.2247449 0.12247449 0.12500000  
## 10 0 0.1 1 1 0.1 1.2247449 0.12247449 0.12500000  
## Label Weight predi Error value adjustment product  
## 1 0 0.08333333 1 1 0.08333333 0.5773503 0.04811252  
## 2 1 0.12500000 1 0 0.00000000 1.7320508 0.21650635  
## 3 1 0.08333333 0 1 0.08333333 0.5773503 0.04811252  
## 4 0 0.08333333 1 1 0.08333333 0.5773503 0.04811252  
## 5 1 0.12500000 0 1 0.12500000 0.5773503 0.07216878  
## 6 1 0.08333333 0 1 0.08333333 0.5773503 0.04811252  
## 7 0 0.08333333 1 1 0.08333333 0.5773503 0.04811252  
## 8 1 0.08333333 0 1 0.08333333 0.5773503 0.04811252  
## 9 0 0.12500000 1 1 0.12500000 0.5773503 0.07216878  
## 10 0 0.12500000 0 0 0.00000000 1.7320508 0.21650635  
## NewWeight  
## 1 0.05555556  
## 2 0.25000000  
## 3 0.05555556  
## 4 0.05555556  
## 5 0.08333333  
## 6 0.05555556  
## 7 0.05555556  
## 8 0.05555556  
## 9 0.08333333  
## 10 0.25000000  
## Label Weight predi Error value adjustment product  
## 1 0 0.05555556 1 1 0.05555556 1.0571883 0.05873268  
## 2 1 0.25000000 1 0 0.00000000 0.9459053 0.23647633  
## 3 1 0.05555556 1 0 0.00000000 0.9459053 0.05255029  
## 4 0 0.05555556 0 0 0.00000000 0.9459053 0.05255029  
## 5 1 0.08333333 0 1 0.08333333 1.0571883 0.08809902  
## 6 1 0.05555556 1 0 0.00000000 0.9459053 0.05255029  
## 7 0 0.05555556 0 0 0.00000000 0.9459053 0.05255029  
## 8 1 0.05555556 1 0 0.00000000 0.9459053 0.05255029  
## 9 0 0.08333333 1 1 0.08333333 1.0571883 0.08809902  
## 10 0 0.25000000 1 1 0.25000000 1.0571883 0.26429707  
## NewWeight  
## 1 0.05882353  
## 2 0.23684211  
## 3 0.05263158  
## 4 0.05263158  
## 5 0.08823529  
## 6 0.05263158  
## 7 0.05263158  
## 8 0.05263158  
## 9 0.08823529  
## 10 0.26470588  
## Label Weight predi Error value adjustment product  
## 1 0 0.05882353 0 0 0.00000000 1.9011275 0.11183103  
## 2 1 0.23684211 0 1 0.23684211 0.5260037 0.12457981  
## 3 1 0.05263158 1 0 0.00000000 1.9011275 0.10005934  
## 4 0 0.05263158 0 0 0.00000000 1.9011275 0.10005934  
## 5 1 0.08823529 0 1 0.08823529 0.5260037 0.04641209  
## 6 1 0.05263158 0 1 0.05263158 0.5260037 0.02768440  
## 7 0 0.05263158 1 1 0.05263158 0.5260037 0.02768440  
## 8 1 0.05263158 1 0 0.00000000 1.9011275 0.10005934  
## 9 0 0.08823529 1 1 0.08823529 0.5260037 0.04641209  
## 10 0 0.26470588 1 1 0.26470588 0.5260037 0.13923626  
## NewWeight  
## 1 0.13571429  
## 2 0.15118577  
## 3 0.12142857  
## 4 0.12142857  
## 5 0.05632411  
## 6 0.03359684  
## 7 0.03359684  
## 8 0.12142857  
## 9 0.05632411  
## 10 0.16897233

sample(0:1,10,replace = T)

> confusionMatrix(table(adadata$Label,adadata$predi))

Confusion Matrix and Statistics

0 1

0 4 1

1 2 3

Accuracy : 0.7

95% CI : (0.3475, 0.9333)

No Information Rate : 0.6

P-Value [Acc > NIR] : 0.3823

Kappa : 0.4

Mcnemar's Test P-Value : 1.0000

Sensitivity : 0.6667

Specificity : 0.7500

Pos Pred Value : 0.8000

Neg Pred Value : 0.6000

Prevalence : 0.6000

Detection Rate : 0.4000

Detection Prevalence : 0.5000

Balanced Accuracy : 0.7083

'Positive' Class : 0

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

Boosting makes multiple trees in series, after finishing the 1st tree then it uses the model to predict on the data which was not selected in 1st iteration, some of the data prediction works well, but for some it doesn’t those data point more likely to be selected during second training set, as long as keep going on each time adjusting model based on Previous prediction and those bad prediction most often come in 2nd prediction, Gradually number of sequences it improves the model.

##########################ROC###########################  
tpr=character()  
fpr=character()  
for(i in seq(0,1,0.01)){#creting sequence of threshold   
   
 tpr=c(tpr,sum(pred\_boost>=i & data2\_test$Credit\_Standing==1)/length(data2\_test$Credit\_Standing==1))  
 fpr=c(fpr,sum(pred\_boost>=i & data2\_test$Credit\_Standing==0)/length(data2\_test$Credit\_Standing==0))  
}

##CREATING TPR ANF FPR VECTOR

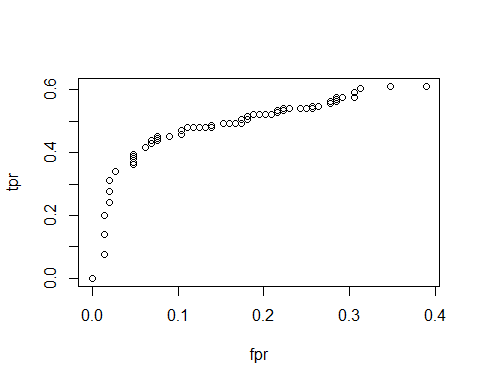
length(tpr)

## [1] 101

length(fpr)

## [1] 101

plot(fpr,tpr)##ploting ROC



The ROC curve is the True positive rate is plotted in function of false positive rate for positive rate for different cut off points.

Each point represents a sensitivity/ specificity pair corresponding to a particular decision threshold.

TPR/Recall/Sensitivity=

Specificity=

FPR=1-specificity

FPR=

Larger the area under the curve better is the accuracy. The area covered in this model seems to be around 75% so here it can be say that the model is fairly good model.

##install.packages(ROCR) #just to verify that ROC with library coming same or not  
library(ROCR)

## Loading required package: gplots

##   
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':  
##   
## lowess

pred=prediction(pred\_boost,data2\_test$Credit\_Standing)  
perf=performance(pred,"tpr","fpr")  
plot(perf)

