Data Mining Assignment 3

TEAM MEMBERS

1. *Nithyadharshni Sampathkumar*
2. *Saai Krishnan Udayakumar*
3. *Sibi Senthur Muthusamy*

#----------

#Loading the Data Set

```
install.packages("xlsx")
library(xlsx)
sample<-read.xlsx(file.choose(), sheetName="Sample for Model Development", header = T)
View(sample)
```

#Here, we are first loading the dataset and are giving the appropriate sheet name to specify the exact sheet from the data frame we are intending to use to build our model,
#Header=True retains the column header exactly as we have it out data frame.

#After, loading our dataset our initial emphasis is on understanding the data type of each attributes,
#We perform the same using str function,

```
str(sample)
head(sample)
```
#From our initial observation we understand that the target variable we want to analyse, is the Manipular
table(sample$Manipulator)
#No (Non-Manipulators) 181
39 (Manipulators)
print(prop.table(table(sample$Manipulator)))

#As we could see we have **82% of instances in the data set belonging to the majority class who are account non-manipulators**
#Slightly, over **17.5% instances belong to the account manipulators section of the class**

(Question #1)

Beneish M-Score is a mathematical model that uses eight financial ratios to identify whether a company has manipulated its earnings. These variables are constructed from the company's financial statements and create a score to describe the degree to which the earnings have been manipulated. The eight variables given together is weighted together according to the formula,

M = -4.84 + 0.92*DSRI + 0.528*GMI + 0.404*AQI + 0.892*SGI + 0.115*DEPI – 0.172*SGAI + 4.679*ACCR – 0.327*LEVI

Based on various selection procedures, the formula was tied down to

M = -6.065 + 0.823*DSRI + 0.906*GMI + 0.593*AQI + 0.717*SGI + 0.107*DEPI

A score greater than -2.22 (i.e. less negative than this) indicates a strong likelihood of a firm being a manipulator

- Beneish used all the companies in the Compustat database between 1982-1992.
- In his out of sample tests, Beneish found that he could correctly identify 76% of manipulators, whilst only incorrectly identifying 17.5% of non-manipulators.
- However, the challenge currently being faced is dealing with an unbalanced data on which the Beneish model has not been tested yet. They have only been tested on certain US financial data, Prowess Database and yet the model has not yet been proven to be successful on all types of data set.

- Hence, we believe model won't be suitable to be tested on Indian data and would need some re-evaluation based on certain type of powerful predictive models.

INFERENCE (Question #2)

#From the table function we infer that we have an **"imbalanced data set"** as the number of account non-manipulators is much greater than the number of account manipulators.
#In such a case, gauging a model's accuracy wouldn't be a good idea, as we are interested in predicting the fewer number of instances of account manipulators.
#Such a scenario would expect us to compute the model's Recall Parameter as an important metric in computing the Model's efficiency we would construct.
#Recall (or hit Rate or sensitivity) is computed as the Ratio of number of instances that are Predicted Positive and are Actually-positive (TP) divided instances which are actually-positive but predicted negative (FN) and instances which are actually postive (TP).

#Recall ~ (TP)/(TP+FN)
#Having an unbalanced data set with observations skewed heavily towards one class (Majority class), the model is likely to generalize the minority class instances into the majority one's which is costlier.
#Our model's accuracy will give us an illusion that our model is performing well, while in true sense we are missing out on classifying a few instances (In our case the account manipulators) which is our primary objective.
#Here our main objective is maximizing our Model's Recall parameter.
#We will handle the imbalanced data set problem through various techniques including Over Sampling, Under Sampling, Combination of both and using the SMOTE technique.`
 # Building our Logistic Regression Model to predict manipulators/Non- Manipulators
(Question #3 and Question #4) (Building Logistic Regression & Accuracy of the Model)

```
sample$Manipulator=factor(sample$Manipulator)
mylogit= glm(Manipulator~DSRI+GMI+AQI+SGI+DEPI+SGAI+ACCR+LEVI, data=sample, family ="binomial")
```

#The argument family is used to describe the error distribution.
#This option is set to binomial, which tells R to perform logistic regression (1 or 0 (two class) classification)
#Since we are doing a logistic regression, the default baseline odds are the odds in the Non-Manipulators category.
#we can change our baseline using the relevel function to the Manipulators category.
#as observed earlier, Recall is of paramount importance to us given the scenario, as we are interested in predicting the fewer number of Account Manipulators who form the minority class.
#also, the cost of misclassifying an account manipulator as a non-manipulator is costlier than classifying an account non-manipulator as a manipulator.
#from, our observation in building our Logistic Regression model, we infer that the models developed below have very high accuracy (Typically ranging between 85% – 90%) irrespective of their construction in the entire data or in test/train data
#this, is because most of the majority class instances are predicted well as the model generalizes them well due to the higher number of occurrences and fewer minority instances leads to a lower Recall value owing to less occurrences in the data set.

```
summary(mylogit)
```

#Logistic Regression coefficients give the change in the log odds of the outcome for a one-unit increase in the predictor variables.
#1 - For one-unit increase in Days Sales to Receivables Index (DSRI), the log odds of Manipulation (versus Non-manipulation) increases by 0.9853

#2 - For one-unit increase in Gross Margin Index (GMI), the log odds of account Manipulation (Versus Non-Manipulation) increases by 1.2394

#Now, we dive into Model evaluation to gauge the accuracy and other evaluation parameters.
#We generate the Confusion Matrix showing different counts including True Positive, False Positive, True Negative, False Negative

sample$pr<-as.vector(ifelse(predict(mylogit,type="response",sample)>0.5,"Yes","No"))

#Generating the table showing Counts,
table(sample$Manipulator, sample$pr, dnn=c("Actual","Predicted"))

#Out of the 39 instances of Companies having, manipulated their Earning Statement, we infer that 19 instances have been misclassified into Non-Manipulated Category.
#This is a typical problem of unbalanced data sets classifying rare events into common events.

*#Measuring our Accuracy Rate ---> **#Accuracy** - (TP+TN)/(P+N) -> **88.18 % (Not a Useful Measure)***
*#Computing Recall (Hit Rate) ---> **#Recall** -- (TP)/(P+N) --> **48.71 % Which needs to be improved significantly to identify fraudulent companies.***

#the, above scenario is a typical instance of a highly unbalanced data set, having high accuracy but lower recall.

#Handling the Class Imbalance Problem?
#We could handle the class imbalance issue with several techniques, most notable are the Under Sampling, Over Sampling, combination of both the techniques with few advantages and disadvantages.

#We use the package **#ROSE (Random Over Sampling Examples)** to generate artificial data based on sampling methods and smoothed bootstrap approach,

a<-sample[,-10]
str(a)
View(a)
install.packages("ROSE")
library("ROSE")

#Let's do both under sampling and over sampling on this imbalanced data to compare results. This can be achieved using **method = "both"**. In this case, the minority class is oversampled with replacement and majority class is under sampled without replacement.

*data_over_under<- ovun.sample (C.MANIPULATOR~., data=a, **method="both"**, N=220, seed = 1)$data*
table(data_balanced_under$C.MANIPULATOR)
##
mylogit=glm(data_over_under$C.MANIPULATOR~.,data=data_over_under, family ="binomial")
data_over_under$prob<-as.vector(ifelse(predict(mylogit,type="response",data_over_under)>0.5,"1","0"))

Model_results<-table(data_over_under$C.MANIPULATOR, data_over_under$prob,dnn=c("Actual","Predicted"))
Model_results
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])
Recall
##100%
#Using the method "both", we get 107 instances as Manipulators against 113 instances as Non-Manipulators.

#Splitting the dataset into training and testing set to understand how the model performs on unseen data,

#Using training and testing data,

```
set.seed(1234)
index<-sample(2, nrow (data_over_under),replace=TRUE, prob= c(0.6,0.4))
training<-data_over_under[index==1,]
testing<-data_over_under[index==2,]
mylogit=glm(training$C.MANIPULATOR~DSRI+GMI+AQI+SGI+DEPI+ACCR, data=training, family ="binomial")
testing$probablity<-as.vector(ifelse(predict(mylogit, type="response", testing)>0.5,"1","0"))
Model_results<-table(testing$C.MANIPULATOR,testing$probablity,dnn=c("Actual","Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1] +Model_results[2,2])
Recall
```

We obtain a recall value of **#88.23529%** through both under sampling and over sampling

#Under Sampling

```
data_balanced_under<- ovun.sample(C.MANIPULATOR ~ ., data=a, method = "under", p=0.5, seed = 1)$data
table(data_balanced_under$C.MANIPULATOR)
```

##Option "under" determines simple under sampling without replacement of the majority class until either the specified sample size N is reached or the positive example has probability p of occurring.

#Running a Logistic Regression on the Under Sampled data,

```
mylogit=glm(data_balanced_under$C.MANIPULATOR~DSRI+GMI+AQI+SGI+DEPI+ACCR,
data=data_balanced_under, family ="binomial")
summary(mylogit)
data_balanced_under$probablity<-
as.vector(ifelse(predict(mylogit,type="response",data_balanced_under)>0.5,"1","0"))
Model_results<-table(data_balanced_under$C.MANIPULATOR,data_balanced_under$probablity,
dnn=c("Actual","Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1] +Model_results[2,2])
```

Recall #82.05%

From Unknown data,

```
set.seed(1200)
index<-sample(2, nrow(data_balanced_under),replace=TRUE, prob=c(0.6,0.4))
training<-data_balanced_under[index==1,]
testing<-data_balanced_under[index==2,]
mylogit=glm(training$C.MANIPULATOR~DSRI+GMI+AQI+SGI+DEPI+ACCR,data=training,family ="binomial")
testing$probablity<-as.vector(ifelse(predict(mylogit,type="response",testing)>0.5,"1","0"))
Model_results<-table(testing$C.MANIPULATOR,testing$probablity,dnn=c("Actual","Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])
```

Recall #85.714%

Inferences

#Under-sampled data frame has almost similar recall - levels on unseen test data as well but it is kind of expected since we are dealing with a small data set.

#Under Sampling aims to randomly eliminate few of the majority class instances, till there is some form of balancing between the majority and minority class instances.

#The major disadvantage of Under Sampling is we run the risk of losing out on some useful information when we build classifiers and we might end up building our model from a biased sample (Resulting in low test error)

#Over Sampling

```
data.bal.ov<- ovun.sample(C.MANIPULATOR ~ ., data = a, method = "over",
                          p = 0.5, seed = 1)$data
table(data.bal.ov$C.MANIPULATOR)
#0 (Non Manipulators)—181 Instances and 1(Manipulators) 166 Instances

mylogit=glm(data.bal.ov$C.MANIPULATOR~DSRI+GMI+AQI+SGI+DEPI+ACCR, data=data.bal.ov, family
="binomial")
summary(mylogit)
data.bal.ov$probablity<-as.vector(ifelse(predict(mylogit,type="response", data.bal.ov)>0.5,"1","0"))
Model_results<-table(data.bal.ov$C.MANIPULATOR, data.bal.ov$probablity, dnn=c("Actual","Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1] +Model_results[2,2])
```

Recall #84.337%

```
set.seed(1983)
index<-sample(2, nrow(data.bal.ov), replace=TRUE, prob=c(0.6,0.4))
training<-data.bal.ov[index==1,]
testing<-data.bal.ov[index==2,]
mylogit=glm(training$C.MANIPULATOR~DSRI+GMI+AQI+SGI+DEPI+ACCR, data=training, family ="binomial")
testing$probablity<-as.vector(ifelse(predict(mylogit, type="response",testing)>0.5,"1","0"))
Model_results<-table(testing$C.MANIPULATOR,testing$probablity,dnn=c("Actual","Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])
```

Recall #88%

#Over Sampling is increasing the number of instances of minority class events by random replication to represent a higher number of minority class instances in the sample.
#One disadvantage would be we run the risk of overfitting as it replicates the minority class events.
#Hence, we apply SMOTE to build our logistic regression model which tries to eliminate few of the disadvantages of Oversampling and Under sampling.

#SMOTE (Recommended Method)

#SMOTE with Logistic Regression

#We try SMOTE to balance our unbalanced dataset (Super Sampling Rare Events ---Synthetic Minority Over-Sampling Technique)
#SMOTE function oversamples rare event using bootstrapping and k-nearest neighbour to synthetically create additional observations of that event.

```
install.packages("unbalanced")
library(unbalanced)
View(sample)
sample$C.MANIPULATOR<-as.factor(as.character(sample$C.MANIPULATOR))
responsevariable2<-sample$C.MANIPULATOR

remVars<-c("Company.ID","Manipulator","C.MANIPULATOR")
remVarsInd <- which(names(sample) %in% remVars)
```

Predictorvariable2<-sample[,-remVarsInd]
View(Predictorvariable2)
str(Predictorvariable2)
#220 observations with 8 variables

#Balancing our unbalanced dataset using SMOTE
data<-ubBalance(X=Predictorvariable2, Y=responsevariable2,type="ubSMOTE",percOver = 250,percUnder = 200,verbose = TRUE)

We have set percOver 250 ----> to arrive at more than double the quantity of Manipulator (Positive) Case
#Setting percUnder 200 ------> to arrive at half the quantity of the Non-Manipulator (Negative) case

balancedData<-cbind(data$X,data$Y)
View(balancedData)
print(prop.table(table(balancedData$`data$Y`)))

#42.857 % (Positive Class Instances) #57.142 % (Negative Class Instances)

#Now Let us fit our linear Model to our Balanced Data with all the variables included,

mylogit=glm(balancedData$`data$Y`~DSRI+GMI+AQI+SGI+DEPI+SGAI+ACCR+LEVI, data=balancedData, family ="binomial")
summary(mylogit)
install.packages("car")
library(car)
vif(mylogit)

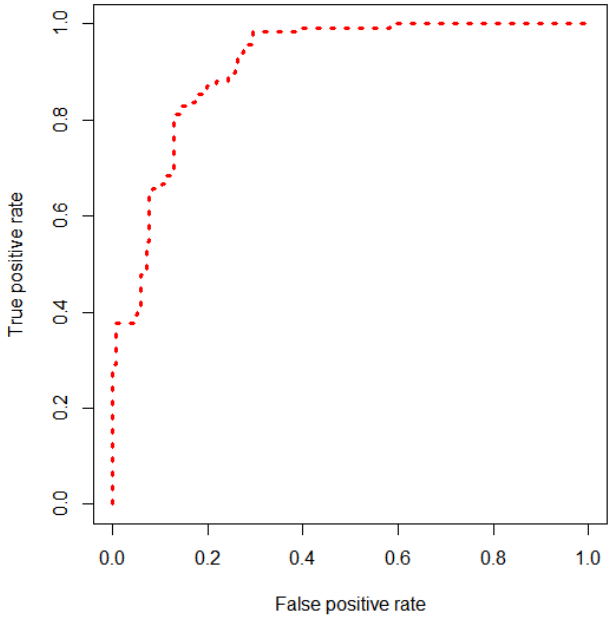
Our Explanatory variables don't seem to be suffering from mulit-collinearity problem as observed using the Variance Inflation Factor(VIF)
#Generally, VIF of above 5 or 10 indicates a Multi Collinearity problem

mylogit=glm(balancedData$`data$Y`~DSRI+GMI+AQI+SGI+DEPI+ACCR,data=balancedData,family ="binomial")
summary(mylogit)
balancedData$probablity<-as.vector(ifelse(predict(mylogit,type="response",balancedData)>0.5,"1","0"))
Model_results<-table(balancedData$`data$Y`,balancedData$probablity, dnn=c("Actual","Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])
Recall

#Recall (~80 % Recall) which is considerably a better accuracy than what the unbalanced data set was giving us!

#Using ROC Curve to set the discrimination threshold,

install.packages("ROCR")
library(ROCR)
fitted.results<-predict(mylogit, type="response",balancedData)
head(fitted.results,5)
pr<-prediction(fitted.results,balancedData$`data$Y`)
perf<-performance(pr,"tpr","fpr")
#plotting the ROC Curve
plot(perf, col = "red", lty = 3, lwd = 3)



#Getting an Optimal Cut-off point,
#In some applications of ROC Curves, we want the point closest to the TPR of one, and FPR of zero. This cut point is "optimal" because it weights both sensitivity and specificity equally,

print(opt.cut(perf,pr))

#We use the optimal cut-off point value to evaluate our sensitivity and specificity

```
opt.cut = function(cost.perf, pred){
  cut.ind = mapply(FUN=function(x, y, p){
    d = (x-0)^2 + (y-1)^2 #Minimum Distance to reach a Sensitivity of 1 with Zero False Alarm Rate
    ind = which(d == min(d))
    c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
      cutoff = p[[ind]])
  }, perf@x.values, perf@y.values, pred@cutoffs)}

print(opt.cut(perf,pr))
```

#We use the optimal cut-off point value to evaluate our sensitivity and specificity

```
cost.perf = performance(pr,"cost")
cost.perf
pr@cutoffs[[1]] [which.min(cost.perf@y.values[[1]])]
```

balancedData$probablity<-as.vector(ifelse(predict(mylogit,type="response",balancedData)>0.**3852873**,"1","0"))
#Using the optimal cut-off point
Model_results<-table(balancedData$`data$Y`,balancedData$probablity, dnn=c("Actual","Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])
Recall
#Recall (~88.88% Recall) with all the variables included which is considerably a better recall-rate than what we obtained with regular cut-offs or in a unbalanced dataset.
summary(mylogit)

#Now, output from out.cut and performance object with measure cost are not equivalent if false positives and false negatives are not weighted equally,
#In our case cost.fn is more expensive than the cost.fp
#Predicting a Non-Manipulator as a Manipulator is less expensive than predicting a Manipulator as a Non-Manipulator,

#Hence, we tune our parameters accordingly,

cost.perf=performance(pr,"cost",cost.fp=1,cost.fn=10)
pr@cutoffs[[1]] [which.min(cost.perf@y.values[[1]])] #We set .1783378 as the cut-off
balancedData$probablity<-as.vector(ifelse(predict(mylogit,type="response",balancedData)>0.1783378,"1","0"))

#Using the optimal cut-off point
Model_results<-table(balancedData$`data$Y`,balancedData$probablity, dnn=c("Actual","Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1] +Model_results[2,2])
Recall
##99.1453

###Splitting into training and testing we get,

set.seed(2001)
index<-sample(2,nrow(balancedData),replace=TRUE,prob=c(0.6,0.4))
training<-balancedData[index==1,]
testing<-balancedData[index==2,]
mylogit=glm(training$`data$Y`~DSRI+GMI+AQI+SGI+DEPI+ACCR+SGAI+LEVI,data=training,family ="binomial")
testing$probablity<-as.vector(ifelse(predict(mylogit,type="response",testing)>0.5,"1","0"))
Model_results<-table(testing$`data$Y`,testing$probablity,dnn=c("Actual","Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])
Recall

#76.7% recall

###Trying to figure out the optimal cut-off point and then evaluating our recall parameter,
fitted.results<-predict(mylogit,type="response",testing)
head(fitted.results,5)
pr<-prediction(fitted.results,testing$`data$Y`)
perf<-performance(pr,"tpr","fpr")
print(opt.cut(perf,pr)) #0.2586402
testing$probablity<-as.vector(ifelse(predict(mylogit,type="response",testing)>0.25864,"1","0"))
Model_results<-table(testing$`data$Y`,testing$probablity,dnn=c("Actual","Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])
Recall #####95.348% recall

#The R function step() (or the stepAIC() function from the MASS package) can be used to perform variable selection.
#Here, we first perform forward selection, we begin by specifying a starting model and the range of models which we want to examine in search.

#Forward selection

null<-glm(balancedData$`data$Y`~DSRI,data=balancedData,family="binomial") #Includes only one variable
full<-glm(balancedData$`data$Y`~DSRI+GMI+AQI+SGI+DEPI+SGAI+ACCR+LEVI,data=balancedData,family ="binomial")
#Includes all the varaibles
step(null,scope=list(lower=null,upper=full),direction="forward")

#This tells R to start with the null model and search through models lying in the range between
#the null and full model using the forward selection algorithm.

#According to the procedure, the best model is the one that includes the variables, DSRI, SGI, ACCR, AQI and GMI (AIC value would be 218.8)
#Notice that AIC(Akaike Information Criteria) is -2log(likelihood)+2p.
#We would want a model selection that gives us with minimum AIC values, here by including the above variables, we get AIC value of 218.8
#We first construct the model with cut-off point as 0.5

```
mylogit=glm(balancedData$`data$Y`~DSRI+GMI+AQI+SGI+ACCR,data=balancedData,family ="binomial")
summary(mylogit)
balancedData$probablity<-as.vector(ifelse(predict(mylogit,type="response",balancedData)>0.5,"1","0"))
Model_results<-table(balancedData$`data$Y`,balancedData$probablity, dnn=c("Actual","Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])
```

Recall ##81.19%

#Using the optimal cut-off point to improve our model performance

```
fitted.results<-predict(mylogit,type="response",balancedData)
pr<-prediction(fitted.results,balancedData$`data$Y`)
perf<-performance(pr,"tpr","fpr")
print(opt.cut(perf,pr)) #0.3464460
balancedData$probablity<-as.vector(ifelse(predict(mylogit,type="response",balancedData)>0.3464460,"1","0"))
Model_results<-table(balancedData$`data$Y`,balancedData$probablity,dnn=c("Actual","Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])
```

Recall ##89.74%

#Using the regular cut-off point to evaluate our model performance on in unseen data,

```
set.seed(2007)
index<-sample(2,nrow(balancedData),replace=TRUE,prob=c(0.6,0.4))
training<-balancedData[index==1,]
testing<-balancedData[index==2,]
mylogit=glm(training$`data$Y`~DSRI+GMI+AQI+SGI+ACCR,data=training,family ="binomial")
summary(mylogit)
testing$probablity<-as.vector(ifelse(predict(mylogit,type="response",testing)>0.5,"1","0"))
Model_results<-table(testing$`data$Y`,testing$probablity,dnn=c("Actual","Predicted"))
Model_results
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])
```

Recall #####82.978% ***Recall for the model with optimal predictor variables, but no-so optimal cut-off point.***

#Model Fit
#After computing recall, we may wish to see how well our model fits
#This can be very useful when comparing competing models,

```
with(mylogit,null.deviance-deviance) #Difference in Deviance
```

##98.16996
#The degrees of freedom for the difference between the two models is equal to the number of predictor variables in the model,

```
with(mylogit,df.null,df.residual)
```

#162
#Finding the p-value,

```
with(mylogit, pchisq(null.deviance-deviance,df.null-df.residual,lower.tail = FALSE))
```

1.284261e-19<0.01 ##TRUE

#The chi-square of 98.16996 with 5 degrees of freedom and an associated p-value of less than 0.01 tells that our model as a whole fits significantly better than an empty model.
#This is sometimes called likelihood ratio test (Deviance residual is -2*log likelihood)

####Predicting the Model performance using the optimal cut-off point,

```
fitted.results<-predict(mylogit,type="response",testing)
head(fitted.results,5)
pr<-prediction(fitted.results,testing$`data$Y`)
perf<-performance(pr,"tpr","fpr")
# Plotting the ROC curve
plot(perf, col ="blue", lty = 3, lwd = 3)
print(opt.cut(perf,pr))  #0.4814687
testing$probablity<-as.vector(ifelse(predict(mylogit,type="response",testing)>0.4814687,"1","0"))
Model_results<-table(testing$`data$Y`,testing$probablity,dnn=c("Actual","Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])
```

Recall ##88.234%

#Further, considering the above model as our baseline logistic regression model we are going to recommend the company,
#We further tune our model, based on the cost factor and we assume that classifying an account-manipulator as a non-manipulator is 5 times costlier than classification the other way around.
#Here, we perform the following code bit to further enhance recall,

```
fitted.results1<-predict(mylogit, type="response", testing)
pr1<-prediction(fitted.results1,testing$`data$Y`)

cost.perf1 = performance(pr1,"cost",cost.fp=1,cost.fn=5)
cost.perf1
pr1@cutoffs[[1]] [which.min(cost.perf1@y.values[[1]])] ##0.16035

testing$probablity<-as.vector(ifelse(predict(mylogit,type="response",testing)>0.16035,"1","0"))
Model_results<-table(testing$`data$Y`,testing$probablity, dnn=c("Actual","Predicted"))
Model_results
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])
Recall ##100%
```

###So, we are essentially telling R to account more strictly the class of manipulators being classified into non-manipulators, and getting a stringent cut-off on which we evaluate our recall
#As a result, our sensitivity will increase, and our false alarm rate will also increase slightly as there is a likelihood that non-manipulators will be classified into manipulators.

```
#We can also perform backward elimination on the same data using the same step command
Step (full, data=balanacedData, direction="backward")

#### Backward elimination includes (selects)all the variables initially and then tries to eliminate variables
step(null, scope=list(upper=full), data=balancedData, direction = "both") #Stepwise regression
```

Summary of the Recall Values,

The below table includes the summary of all the recall values obtained using various techniques of unbalanced data set. We have employed Under Sampling, Over Sampling, SMOTE to deal with unbalanced dataset. At every instance, we have used our entire data and split our data as training and testing to construct our model. We have also employed optimal cut-off point & Cost based threshold (Evaluating False Negatives as costlier than

False Positives) above which classification will be determined based on the ROC curve threshold point where we strived to maximize Sensitivity and Specificity.

Technique	Data Used	Cut-off Point	Recall Values
Both (Over & Under sampling)	Entire Data	0.5	100%
Both (Over & Under sampling)	Training and testing	0.5	88.235%
Under Sampling	Entire Data	0.5	82.05%
Over Sampling	Entire Data	0.5	84%
Under Sampling	Training and testing	0.5	85.714%
Over Sampling	Training and testing	0.5	88%
Smote	Entire Data	0.5	79.487% (~80%)
Smote	Entire Data	0.385 (Optimal Cut-off using ROC)	88.88%
Smote	Entire Data	0.1783378 (Cost-Cut off)	99.1453%
Smote	Training and Testing	0.5	76.7%
Smote	Training and Testing	0.25864 (Optimal using ROC)	95.348%
Smote (Forward Selection)	Entire Data	0.5	81.19%
Smote (Forward Selection)	Entire Data	0.34644 (Optimal Cut-off)	89.74%
Smote (Forward Selection)	Training and Testing	0.5	82.978%
Smote (Forward Selection)	**Training and Testing**	**0.4814687**	**87.234%**
Smote (Forward Selection)	Training and Testing	0.16035 (Optimal Cost Cut-off)	100%

Cross Validated Average Recall is ~ 80.85%

#Performing 10-fold cross validation and trying to compute the average recall value,

#We specify the value of "k" to split the data set into equal sizes with each observation used once in testing and once in training
#Create 10 equally size folds

```
folds <- cut(seq(1, nrow(balancedData)),breaks=10,labels=FALSE)
folds
#Perform 10-fold cross validation
for(i in 1:10) {
  #Segment your data by fold using the which() function
  testIndexes <- which(folds==i, arr.ind=TRUE)
  #Here we index the first of (k) randomly sampled rows and the index is fed to the data frame for getting our test data
  testData <- balancedData[testIndexes,] #Validation Set
  #The remaining (k-1) folds are in the trainData, we repeat the loop for the total "k" values
  trainData <- balancedData[-testIndexes,]

  mylogit=glm(trainData$`data$Y`~DSRI+GMI+AQI+SGI+DEPI+ACCR,data=trainData,family ="binomial")
  testData$probablity<-as.vector(ifelse(predict(mylogit,type="response",testData)>0.48146,"1","0"))
  Model_results<-table(testData$`data$Y`, testData$probablity, dnn=c("Actual","Predicted"))

  Recall[i]<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])*100
  print(Recall[i])
}
```

#60% #54.5454% #90% #83.33% #81.818% #90.909% #100% #56.25% #100% #91.67% (Recall values of each fold)

Average Recall is 80.85%

#Question 5&6 (We would suggest MCA Tech to use the second last model in the table)

We would suggest MCA technologies to make sure to cater to the unbalanced dataset by using SMOTE method and balance the data set, so that the data would be trained to identify the manipulator instances and avoid misclassifying them into non-manipulator instances. Also, MCA would do well to get the optimal cut-point to maximize the Sensitivity and Specificity values for minimum false positive rate. Based on the technique, **we obtained a Recall value of around 88 %** in an unknown test data set.

So, MCA would benefit from using such a Logistic Regression Model **with optimal Cut-point at 0.48** which maximizes Sensitivity and Specificity using SMOTE. Also, employing the model on 10-fold cross validation we obtained an average recall value of around 81%.

Gauging the Manipulator Score,

For our Second Last Model, we write our Logistic Regression Model as,

Manipulator Score = Log(P=Manipulator/(1-P=Manipulator)) = -9.7220+1.4315(DSRI)+1.1501(GMI)+0.9117(AQI)+3.9086(SGI)+8.990(ACCR)

We already knew from our previously constructed Logistic Regression that a thresh-hold of 0.48 maximizes our Recall and Sensitivity values for zero false positive rate.

Using the cut-off score of 0.48 as probability threshold, taking exponentiation on both sides of the Logistic Regression equation and evaluating the equation,
We get a Manipulator Score (M) of (-) 0.08 for our unbalanced data set handled via SMOTE.
A score greater than (-) 0.08 indicates a greater likelihood that a company is manipulating.

Note that we can obtain different values of M-score for different values cut-off point we obtained to build our logistic regression model. However, we are using the second last model to construct our Manipulator Score.

#Question 8 (Performing Logistic Regression using the entire data set)

```
install.packages("xlsx")
library(xlsx)
sample<-read.xlsx (file.choose (), sheetName="Complete Data", header = T)
View(sample)
```

#Here, we are first loading the dataset and give the sheet name to specify the exact sheet from the data frame we are aspiring to download
#Header=True retains each column header exactly as we have it out data frame.

#After, loading our dataset our initial emphasis is on understanding the data type of each attributes,
#We perform the same using str function,

```
str(sample)
head(sample)
```

#From our initial observation we understand that the target variable we want to analyse, is the Manipulater,

table(sample$Manipulater)
#No (Non-Manipulators) 1200 # 39 (Manipulators)
print(prop.table(table(sample$Manipulater)))
#as we could see we have 96.85% of instances in the data set belonging to the majority class who are account non-manipulators.
#slightly, **over 3.14% instances belong to the account manipulators section of the class.**

#From the table function we infer that we have an imbalanced data set as the number of account non-manipulators is much greater than the number of account manipulators
#In such a case, gauging a model's accuracy wouldn't be a good idea, as we are interested in predicting the fewer number of instances of account manipulators.
#Such a scenario would expect us to compute the model's Recall parameter as an important metric in computing the Model's efficiency we would construct.

#Recall (or hit Rate or sensitivity) is computed as the Ratio of number of instances that are Predicted Positive and are positive (TP) divided instances which are positive but predicted negative (FN)
#Recall ~ (TP)/(TP+FN)

#Having an unbalanced data set with observations skewed heavily towards one class (Majority class), the model is likely to generalize the minority class instances into the majority one's.
#Our model's accuracy will give us an illusion that our model is performing well, while in true sense we are missing out on classifying a few instances (In our case the account manipulators) which is our primary objective.
#Here our main objective is maximizing our Model's Recall parameter.

 # Building our Logistic Regression Model to predict manipulators/Non- Manipulators
sample$Manipulater=factor(sample$Manipulater)
mylogit=glm(Manipulater~DSRI+GMI+AQI+SGI+DEPI+SGAI+ACCR+LEVI, data=sample, family ="binomial")

#The argument family is used to describe the error distribution.
#This option is set to binomial, which tells R to perform logistic regression (1 or 0 (two class) classification)
#Since we are doing a logistic regression, the default baseline odds are the odds in the Non-Manipulators category.
#We can change our baseline using the relevel function to the Manipulators category.

summary(mylogit)
#Logistic Regression coefficients give the change in the log odds of the outcome for a one-unit increase in the predictor variables.
#1 - For one unit increase in Days Sales to Receivables Index (DSRI), the log odds of Manipulation (versus Non-manipulation) increases by 0.647257
#2 - For one unit increase in Gross Margin Index (GMI), the log odds of account Manipulation (Versus Non-Manipulation) increases by 0.495750

#Now, we dive into Model evaluation to gauge the accuracy and other evaluation parameters.
#We generate the Confusion Matrix showing different counts including True Positive, False Positive, True Negative, False Negative

sample$pr<-as.vector(ifelse(predict(mylogit, type="response",sample)>0.5,"Yes","No"))

#Generating the confusion Matrix showing Counts,
table(sample$Manipulater,sample$pr, dnn=c("Actual","Predicted"))

#Out of the 39 instances of Companies having, Manipulated their Earning Statement, we infer that 28 instances have been misclassified into Non-Manipulated Category.

#This is a typical problem of unbalanced data sets classifying rare events into common events.

#Measuring our Accuracy Rate ---> #Accuracy - (TP+TN)/(P+N) -> **97.49% (Not a Useful Measure)**
#Computing Recall (Hit Rate) ---> #Recall -- (TP)/(P+N) --> **28.20 %** Which needs to be improved to identify fraudulent companies.

#the above evaluation metric values are illustrations of typical unbalanced data set, case of very high accuracy but lower recall values.

#For Unbalanced Data Set # <Based on Professor's Reading>

#SMOTE with Logistic Regression
#We try SMOTE to balance our unbalanced dataset (Super Sampling Rare Events ---Synthetic Minority Over-Sampling Technique)
#SMOTE function oversamples rare event using bootstrapping and k-nearest neighbour to synthetically create additional observations of that event

```
install.packages("unbalanced")
library(unbalanced)
install.packages("xlsx")
library(xlsx)
df<-a
df<-read.xlsx(file.choose(), sheetName="Complete Data", header = T)
print(prop.table(table(df$C.MANIPULATOR)))
###Only 3.1% of the observations are manipulators, while almost 97% of observations are non-manipulators
str(df)
View(df)
df$C.MANIPULATOR<-as.factor(as.character(df$C.MANIPULATOR))
df1<-df
response<-df$C.MANIPULATOR
str(response)
View(response)

remVars<-c("Company.ID","Manipulater","C.MANIPULATOR")
remVarsInd <- which(names(df) %in% remVars)
Predictor<-df[,-remVarsInd]
View(Predictor)
str(Predictor)
```
#1239 observations with 8 variables

#Balancing our unbalanced dataset using SMOTE
data1<-ubBalance(X=Predictor, Y=response, type="ubSMOTE", percOver = 1200, percUnder = 150, verbose = TRUE)
# We have set percOver to 1200 ----> to balance out Manipulator quantities at almost the same size of Non-Manipulator Case
#Setting percUnder to 150 ------> to arrive at almost half the quantity of the Non-Manipulator case
#Proportion of Positive after ubSMOTE: 41.94 % of 1239 observations

```
data1
str(data1)

balancedData<-cbind (data1$X, data1$Y)
View(balancedData)
Print (prop.table (table(balancedData$`data1$Y`)))
#41.935 % (Positive Class Instances) #58.064 % (Negative Class Instances)

#Now Let us fit our linear Model to our Balanced Data
#With all the variables included,
install.packages("car") #For checking multicollinearity issues
library(car)

mylogit=glm(balancedData$`data1$Y`~DSRI+GMI+AQI+SGI+DEPI+SGAI+ACCR+LEVI,data=balancedData,fa
mily ="binomial")
vif(mylogit)
#Our Explanatory variables don't seem to be suffering from Multi collinearity problem as observed using
the Variance Inflation Factor(VIF)
#Generally, VIF values of above 5 or 10 indicates a Multi Collinearity problem
balancedData$probablity<-as.vector (ifelse(predict(mylogit,type="response",balancedData)>0.5,"1","0"))
#Using the optimal cut-off point
Model_results<-table(balancedData$`data1$Y`,balancedData$probablity, dnn=c("Actual", "Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])

Recall ##77.289%

#Using ROC Curve to set the discrimination threshold,
install.packages("ROCR")
library(ROCR)
fitted.results<-predict(mylogit,type="response", balancedData)
head(fitted.results,5)
pr<-prediction(fitted.results,balancedData$`data1$Y`)
perf<-performance(pr,"tpr","fpr")
#plotting the ROC Curve
plot(perf, col = "red", lty = 3, lwd = 3)

#Getting an Optimal Cut-off point,
#In some applications of ROC Curves, we want the point closest to the TPR of one, and FPR of zero. This
cut point is "optimal" because it weights both sensitivity and specificity equally,

opt.cut = function(cost.perf, pred){
  cut.ind = mapply(FUN=function(x, y, p){
    d = (x-0)^2 + (y-1)^2
    ind = which(d == min(d))
    c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
      cutoff = p[[ind]])
  }, perf@x.values, perf@y.values, pred@cutoffs)}

print(opt.cut(perf,pr))
#We use the optimal cut-off point value to evaluate our sensitivity and specificity
```

balancedData$probablity<-
as.vector(ifelse(predict(mylogit,type="response",balancedData)>0.3493464,"1","0")) #Using the optimal cut-off point
Model_results<-table(balancedData$`data1$Y`,balancedData$probablity, dnn=c("Actual","Predicted"))
Model_results
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])
Recall

#Recall (~90.729% Recall) with all the variables included which is considerably a better recall-rate than what we obtained with regular cut-offs or in a unbalanced dataset
summary(mylogit)
#Now, output from out.cut and performance object with measure cost are not equivalent if false positives and false negatives are not weighted equally,
#In our case cost.fn is more than the cost.fp
#Predicting a Non-Manipulator as a Manipulator is less expensive than predicting a Manipulator as a Non-Manipulator,

#Hence, we tune our parameters accordingly,

cost.perf=performance(pr,"cost",cost.fp=1,cost.fn=10)
pr@cutoffs[[1]] [which.min(cost.perf@y.values[[1]])] #We set .1663778 as the cut-off
balancedData$probablity<-
as.vector(ifelse(predict(mylogit,type="response",balancedData)>0.1663778,"1","0")) #Using the optimal cut-off point
Model_results<-table(balancedData$`data1$Y`,balancedData$probablity, dnn=c("Actual","Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])

Recall ##98.61933

###Splitting into training and testing we get,
set.seed(2001)
index<-sample(2,nrow(balancedData),replace=TRUE, prob=c(0.6,0.4))
training<-balancedData[index==1,]
testing<-balancedData[index==2,]
mylogit=glm(training$`data1$Y`~DSRI+GMI+AQI+SGI+DEPI+ACCR+SGAI+LEVI,data=training,family ="binomial")
testing$probablity<-as.vector(ifelse(predict(mylogit,type="response",testing)>0.5,"1","0"))
Model_results<-table(testing$`data1$Y`,testing$probablity,dnn=c("Actual","Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])

Recall #80.88% recall

###Trying to figure out the optimal cut-off point and then evaluating our recall parameter,
fitted.results <-predict (mylogit, type="response", testing)
head(fitted.results,5)
pr<-prediction(fitted.results,testing$`data1$Y`)
perf<-performance(pr,"tpr","fpr")
*print(opt.cut(perf, pr)) **#0.3824126***
testing$probablity<-as.vector(ifelse(predict(mylogit, type="response", testing)>0.3824126,"1","0"))
Model_results<-table(testing$`data1$Y`,testing$probablity, dnn=c("Actual", "Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])

Recall #90.196% recall

#The R function step() (or the stepAIC() function from the MASS package) can be used to perform variable selection.
#Here, we first perform forward selection, we begin by specifying a starting model and the range of models which we want to examine in search.

#Forward selection
null<-glm(balancedData$`data1$Y`~DSRI, data=balancedData,family="binomial") #Includes only one variable
full<-glm (balancedData$`data1$Y`~DSRI+GMI+AQI+SGI+DEPI+SGAI+ACCR+LEVI, data=balancedData, family ="binomial")
#Includes all the variables
Step (null, scope=list (lower=null, upper=full), direction="forward")
#This tells R to start with the null model and search through models lying in the range between
#the null and full model using the forward selection algorithm.

#According to the procedure, the best model is the one that includes the variables, DSRI, SGI, ACCR, AQI, GMI, DEPI, LEVI (AIC value would be 910.63)
#Notice that AIC (Akaike Information Criteria) is -2log(likelihood)+2p.
#We would want a model selection that gives us with minimum AIC values, here by including the above variables, we get AIC value of 910.63
#We first construct the model with cut-off point as 0.5

mylogit=glm(balancedData$`data1$Y`~DSRI+GMI+AQI+SGI+ACCR+LEVI+DEPI,data=balancedData,family ="binomial")
summary(mylogit)
balancedData$probablity<-as.vector(ifelse(predict(mylogit,type="response",balancedData)>0.5,"1","0"))
Model_results<-table(balancedData$`data1$Y`,balancedData$probablity, dnn=c("Actual","Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])
Recall ##78%

#Using the optimal cut-off point to improve our model performance
fitted.results<-predict(mylogit,type="response", balancedData)
pr<-prediction(fitted.results, balancedData$`data1$Y`)
perf<-performance(pr,"tpr","fpr")
print(opt.cut(perf,pr)) #0.4097292
balancedData$probablity<-
as.vector(ifelse(predict(mylogit,type="response",balancedData)>0.4097292,"1","0"))
Model_results<-table(balancedData$`data1$Y`,balancedData$probablity,dnn=c("Actual","Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])
Recall ##86.39053%

#Using the regular cut-off point to improve our model performance on in unseen data,
set.seed(2007)
index<-sample(2,nrow(balancedData),replace=TRUE, prob=c(0.6,0.4))
training<-balancedData[index==1,]
testing<-balancedData[index==2,]
mylogit=glm(training$`data1$Y`~DSRI+GMI+AQI+SGI+ACCR+LEVI+DEPI,data=training,family ="binomial")
summary(mylogit)
testing$probablity<-as.vector(ifelse(predict(mylogit,type="response",testing)>0.5,"1","0"))
Model_results<-table(testing$`data1$Y`,testing$probablity,dnn=c("Actual","Predicted"))

```
Model_results
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])
```
Recall #####74.8718% Recall for the model with optimal predictor variables, but no-so optimal cut-off point.

```
#After computing recall, we may wish to see how well our model fits
#This can be very useful when comparing competing models,
with(mylogit, null.deviance -deviance) #Difference in Deviance
##432.8595
#The degrees of freedom for the difference between the two models is equal to the number of predictor variables in the model,
with(mylogit, df.null, df.residual)
#716
#Finding the p-value,
with(mylogit,pchisq(null.deviance-deviance,df.null-df.residual,lower.tail = FALSE))
2.125699e-89<0.01 ##TRUE
#Chi-square of 432.8595 and an associated p-value of less than 0.01 tells that our model as a whole fits significantly better than an empty model.
#This is sometimes called likelihood ratio test (Deviance residual is -2*log likelihood)

#Predicting the Model performance using the optimal cut-off point,
fitted.results<-predict(mylogit, type="response",testing)
head(fitted.results,5)
pr<-prediction(fitted.results,testing$`data1$Y`)
perf<-performance(pr,"tpr","fpr")
# Plotting the ROC curve
plot(perf, col ="blue", lty = 3, lwd = 3)
print(opt.cut(perf,pr)) #0.3900214
testing$probablity<-as.vector(ifelse(predict(mylogit,type="response",testing)>0.3900214,"1","0"))
Model_results<-table(testing$`data1$Y`,testing$probablity,dnn=c("Actual","Predicted"))
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])
```
Recall ##88.717%

```
#Further, considering the above model as our baseline logistic regression model we are going to recommend the company,
#We further tune our model, based the cost factor and we assume that classifying a account-manipulator as a non-manipulator is 5 times more costly than classification the other way round.
#Here, we perform the following code bit to further enhance recall,
fitted.results1<-predict(mylogit,type="response",testing)
pr1<-prediction(fitted.results1,testing$`data1$Y`)

cost.perf1 = performance(pr1,"cost",cost.fp=1,cost.fn=5)
cost.perf1
pr1@cutoffs[[1]] [which.min(cost.perf1@y.values[[1]])] ##0.2790819

testing$probablity<-as.vector(ifelse(predict(mylogit, type="response",testing)>0.2790819,"1","0"))
Model_results<-table(testing$`data1$Y`, testing$probablity, dnn=c("Actual","Predicted"))
Model_results
Recall<-(Model_results[2,2])/(Model_results[2,1]+Model_results[2,2])
```
Recall ##95.897%

###So, we are essentially telling R to account more strictly the class of manipulators being classified into non-manipulators, and getting a stringent cut-off on which we evaluate our recall parameter
#As a result, our sensitivity will increase, and our false alarm rate will also increase slightly as there is a likelihood that non-manipulators will be classified into manipulators.

#We can also perform backward elimination on the same data using the same step command
Step (full, data=balanacedData, direction="backward") #### Backward elimination includes (selects)all the variables initially and then tries to eliminate variables
step (null, scope=list(upper=full), data=balancedData, direction = "both") #Stepwise regression

Below table shows the consolidated Recall values for different Logistic Regression model constructed using the entire data set,

Technique	Data	Cut-off	Recall
Smote	Entire Data	0.5	77.289%
Smote	Entire Data	0.3493464 (Optimal Cut-off using ROC)	90.73%
Smote	Entire Data	0.1663778(Cost-Cut off)	98.62%
Smote	Training and Testing	0.5	80.88%
Smote	Training and Testing	0.3824126 (Optimal using ROC)	90.20%
Smote (Forward Selection)	Entire Data	0.5	78%
Smote (Forward Selection)	Entire Data	0.4097292 (Optimal Cut-off)	86.39%
Smote (Forward Selection)	Training and Testing	0.5	74.88%
Smote (Forward Selection)	**Training and Testing**	**0.3900214** (Optimal Cut-off)	**88.72%**
Smote (Forward Selection)	Training and Testing	0.2790819 (Optimal Cost Cut-off)	95.89%

Constructing a logistic regression model on our entire data set, we observe that our model performs almost similar in terms of accuracy as compared to our previous model but slightly varying Recall values, except that the testing error increases a bit when constructing a model with a larger data set. This may be because the model must learn new patterns from the training set and is unable to generalise the same into an unknown data set.

Decision Tree, Random Forest, Boosting (Question 7 and Question 9)

#Developing the Cartesian and Regression Tree Model (CART Tree)
#Construct a decision tree model using rpart() from "rpart" package

```
install.packages("rpart")
library(rpart)
#We are building the decision tree using the balanced data we used to run the Logistic Regression
str(balancedData)
#273 observations having 9 columns
balancedData<-balancedData[,-10]
str(balancedData)

#First, we are trying to predict a default decision tree using the rpart package
View(balancedData)

manipulator_rpart = rpart (balancedData$`data$Y`~., data = balancedData, method = "class",
parms=list(split="information"))

# To plot rpart decision tree we can either use rpar.plot() function from rpart.plot package or fancyRpartPlot()
function from "rattle" package.
```

```
library(rpart.plot)
rpart.plot(manipulator_rpart)
print(manipulator_rpart)
manipulator_rpart$variable.importance

#Our initial analysis after constructing the tree would be some of the variables of importance,
#In a decision tree, variables that appear at the top are considered important,
#Hence in our case, ACCR, DSRI, SGI are deemed to be important variables.
#Our variable importance parameter also explains the same.

manipulator_rpart_predict<-predict(manipulator_rpart, newdata = balancedData, type="class")
table_predict<-table(manipulator_rpart_predict,balancedData$`data$Y`,dnn=c("predicted","actual"))
recall<-(table_predict[2,2])/(table_predict[2,2]+table_predict[2,1])
```

Recall #92%

```
#Using the default decision tree, we get a good Recall value, now we try to build a decision tree using optimal value
of (Complexity Parameter) cp, and rpart. control parameters.
#Let gauge, the Accuracy as appropriate

build_rpart<- rpart(balancedData$`data$Y`~., data=balancedData, method="class",
control=rpart.control(minsplit=10),parms=list(split="information"))

#To construct a pruned tree with the minimum prediction error we choose the optimal complexity parameter using
the CP table,

opt<-which.min(manipulator_rpart$cptable[,"xerror"])
cp<-manipulator_rpart$cptable[opt,"CP"]
cp
build_prune<-prune(build_rpart, cp=cp)
print(build_prune)
manipulator_rpart_predict<-predict(build_prune,newdata = balancedData,type="class")
table_predict<-table(manipulator_rpart_predict , balancedData$`data$Y`, dnn=c("predicted","actual"))
table_predict
recall<-(table_predict[2,2])/(table_predict[2,2]+table_predict[2,1])
recall
```

#92.62% (Recall is around 92.62%)

```
set.seed(1908)
ind<-sample(2, nrow(balancedData), replace=TRUE, prob=c(0.7,0.3))
training<-balancedData[ind==1,]
testing<-balancedData[ind==2,]

build_rpart<- rpart(training$`data$Y`~., data=training,
method="class",control=rpart.control(minsplit=10),parms=list(split="information"))

#To construct a pruned tree with the minimum prediction error we choose the optimal complexity
parameter using the cp table,

opt<-which.min(build_rpart$cptable[,"xerror"])
cp<-build_rpart$cptable[opt,"CP"]
cp
build_prune<-prune(build_rpart,cp=cp)
```

```
print(build_prune)
manipulator_rpart_predict<-predict(build_prune,newdata = testing, type="class")
table_predict<-table(manipulator_rpart_predict, testing$`data$Y`,dnn=c("predicted","actual"))
table_predict
recall<-(table_predict[2,2])/(table_predict[2,2]+table_predict[2,1])
recall
```

#81.25%

##RandomForest

```
install.packages("randomForest")
library(randomForest)
```

##RandomForest
##Splitting the data into training and testing, RandomForest cannot handle missing values and in our data set we don't have any missing values
#Checking again if there are any missing values,
colSums(is.na(balancedData)) #We don't find any missing values,

```
set.seed (1908)
ind<-sample (2, nrow(balancedData),replace=TRUE, prob=c(0.7,0.3))
training<-balancedData[ind==1,]
testing<-balancedData[ind==2,]
```

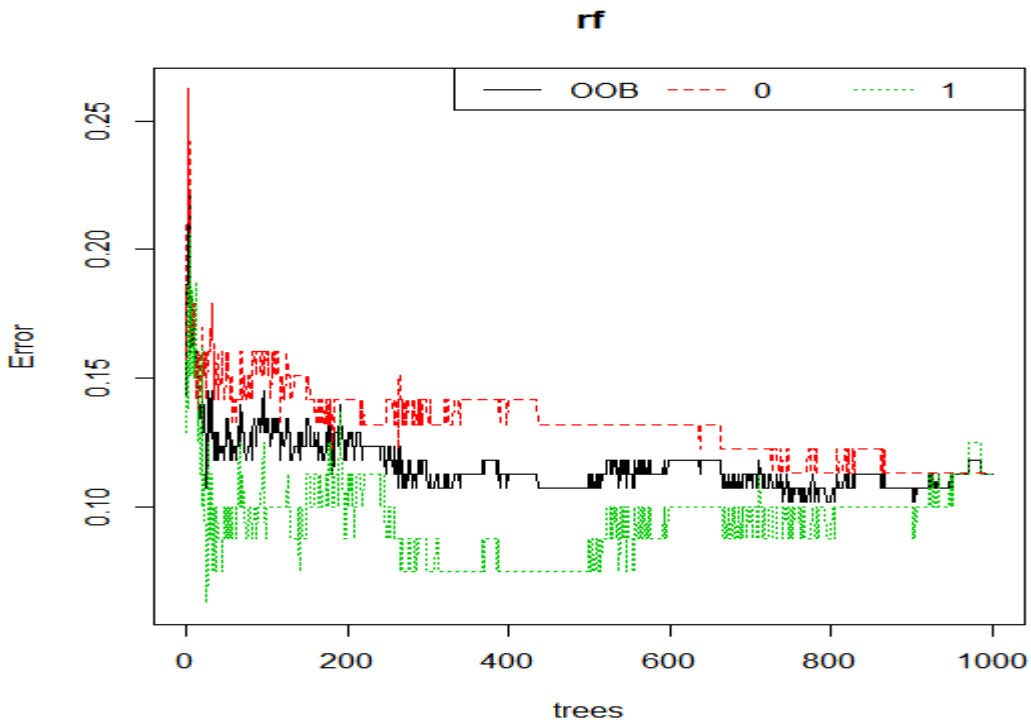
###Building the Random Forest model,

```
rf= randomForest(training$`data$Y`~.,data=training,ntree=1000,proximity=TRUE,replace=TRUE,
        samplesize=ceiling(0.65*nrow(training)), importance=TRUE, mtry=sqrt(ncol(training)))
```

str(training)

#When we set the training set for the current tree with replacement, about 1/3rd (35%) of the cases are left out of the sample.
#The oob(out-of-bag) data is used to get a running unbiased estimate of the classification error as trees are added to the forest. It is also used to get estimates of variable importance.
#ntree is the number of trees to grow
#mtry is the number of variables randomly sampled in each decision tree
##Keeping mtry as the number of variables in the training data will likely lead to co-relation between the trees and hence we optimize by reducing the count,

```
print(rf)
#No. of variables tried at each split: 3 (mtry)
#OOB estimate of error rate: 9.68% (Out of Bag Error)
plot(rf)
legend("topright", legend=colnames(rf$err.rate),lty=c(1,2,3,4),col=c(1,2,3,4),horiz=T)
attributes(rf)
```



importance(rf) #Mean Decrease Accuracy decreases when we drop that variable
varImpPlot(rf) #We infer that DSRI and SGI are the most important variables

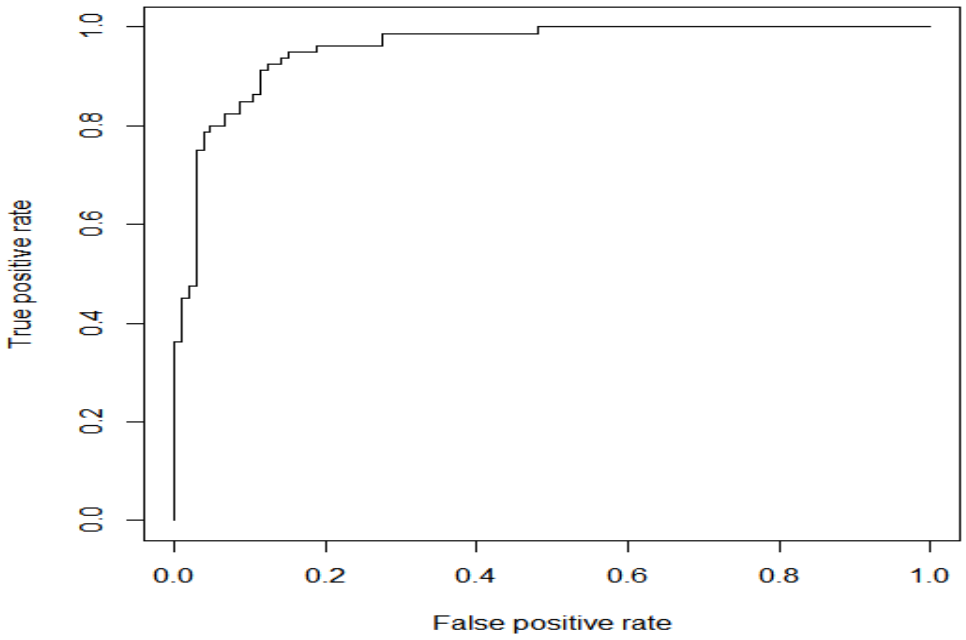
#Predicted values at each instances,
rf$predicted
rf$votes
#Votes are the average predicted values for each instance (1/0) across all the 1000 decision trees constructed with a given sample size

#Using the test data to predict the instance,
##Get Accuracy of Prediction on test data using the predict function,
pred<-predict (rf, newdata = testing)
pred
table_predict<-table(testing$`data$Y`,pred,dnn=c("Actual","Predicted")) #Actual vs Predicted values
recall<-table_predict[2,2]/(table_predict[2,2] +table_predict[2,1])

Recall ##94.59%

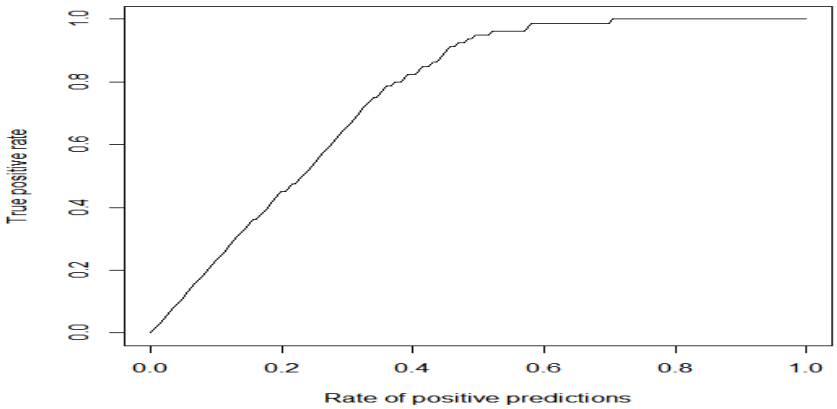
##Adding the various Evaluation Charts for Understanding,
install.packages("ROCR")
library(ROCR)
rf$votes #Using the rf votes to construct the various evaluation charts
pred.pr<-as.data.frame(rf$votes)$'1'
pred<-prediction(pred.pr,training$`data$Y`) ##Evaluating various chart options using the prediction function

##ROC Curve
pred.roc<-performance(pred,"tpr","fpr")
plot(pred.roc)



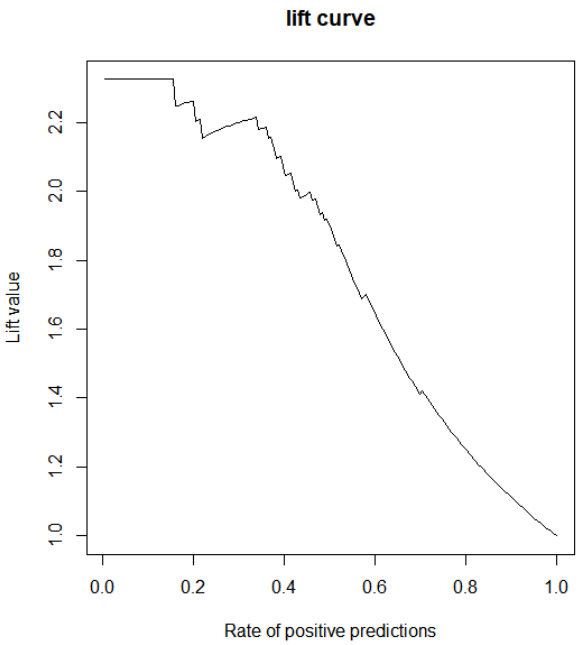
##Gain Chart

pred.gain<-performance(pred,"tpr","rpp") ###True Positive Rate versus Rate of Positive Predictions
plot(pred.gain)



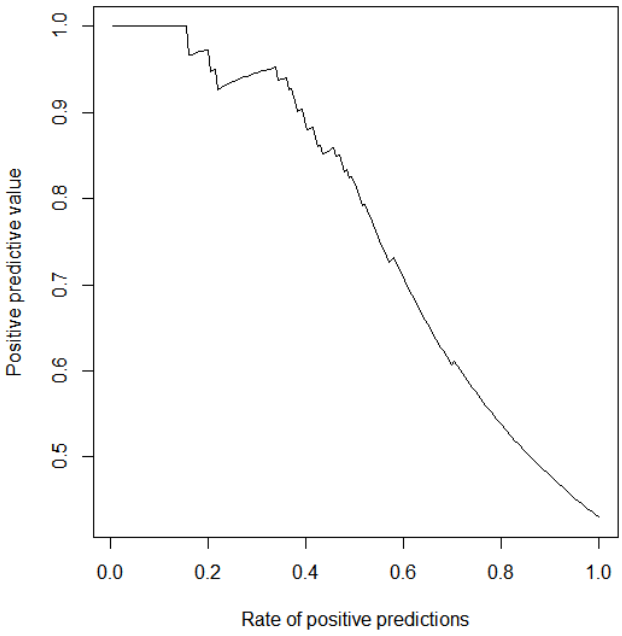
##Lift Chart

#Lift is a measure of the effectiveness of the predictive model calculated as the ratio between the results obtained with and without the predictive model,
per<-performance(pred,"lift","rpp")
plot (per, main="lift curve")



##Response Chart

pred.res<-performance(pred,"ppv","rpp")
plot(pred.res)



##Boosting

##Adabag
install.packages("mboost")
library(mboost)
install.packages("adabag")
library(adabag)
install.packages("party")
View(balancedData)
str(balancedData)
balancedData$`data$Y`<-as.numeric(as.character(balancedData$`data$Y`))

```
set.seed(1234)
ind<-sample(2, nrow(balancedData),replace=TRUE, prob=c(0.6,0.4))
training<-balancedData[ind==1,]
testing<-balancedData[ind==2,]
str(training)

## fit logistic regression model via gradient boosting
manipulator.boost<-glmboost(training$`data$Y`~.,data=training)
manipulator.boost
coef(manipulator.boost, off2int=TRUE)

install.packages("ROCR")
library(ROCR)

b<-prediction(a,testing$`data$Y`)
c<-performance(b,"tpr","fpr")
print(opt.cut(c, b)) #0.312

testing$probablity<-ifelse(predict(manipulator.boost,newdata=testing,type="response")>0.3,"1","0")
View(testing)
pred.table<-table(testing$`data$Y`,testing$probablity,dnn=c("Actual","Predicted"))
pred.table
pred.table_recall<-pred.table[2,2]/(pred.table[2,1]+pred.table[2,2])
pred.table_recall  #93.18%
```

Model	Data	Recall
Random Forest	Training and Testing	94.59%
Boosting	Training and Testing	**93.18%**
Decision Trees	Training and Testing	**81.25%**

#From the table we can infer that, Random Forest which works by constructing multitude of decision trees outperforms Decision Trees in Recall Estimate on unseen data.

Question #10

We have constructed our model using logistic regression and ensemble methods to predict account manipulators and non-manipulators. We have also constructed the model using entire data or using test/train data at various steps. Following our model construction, we infer that Random Forest gives higher Recall value which is our paramount evaluation parameter considering our scenario. We understand that our model performs very well with Logistic Regression with optimal cut-off point, but even more so with ensemble methods like Random Forest. Hence, we would recommend the firm to try Random Forest as priority, and then Logistic Regression to predict earning manipulators and non-manipulators. But for companies having multiple ''NA'' values (missing values) and more levels of categorical value in their financial data, Random Forest will be a difficult method to implement. Such scenarios would call for using Logistic Regression as the evaluation parameter.