Predicting Earning Manipulation- Case Study

Varun Maheshwari

1. The case study deals with classifying organizations with the given 8 metrics as either manipulator or non- manipulator.
   While identifying manipulators is important, it should not be done at a high cost of misclassifying non-manipulators as manipulators; in order to prevent wastage of resources.

   Reading data:

```
library(readxl)
full_data <- read_excel("Desktop/IMB579-XLS-ENG.xlsx", sheet = "Complete Data")
View(full_data)
str(full_data)
```

   Renaming target variable and converting into factor:

```
library(dplyr)            #To use pipeline function to rename
fulldata <- fulldata %>% rename(target = `C-MANIPULATOR` )
str(fulldata)
full_data$`C-MANIPULATOR` <- as.factor(full_data$`C-MANIPULATOR`)
fulldata <- subset(full_data, select = -c(`Company ID`, Manipulater))
rm(full_data)
```

   Treating the data:

```
library(funModelling)
df_status(fulldata)
```

```
> df_status(fulldata)
  variable q_zeros p_zeros q_na p_na q_inf p_inf    type unique
1     DSRI       1    0.08    0    0     0     0 numeric   1224
2      GMI       0    0.00    0    0     0     0 numeric   1148
3      AQI       0    0.00    0    0     0     0 numeric   1238
4      SGI       0    0.00    0    0     0     0 numeric   1239
5     DEPI       0    0.00    0    0     0     0 numeric   1227
6     SGAI       4    0.32    0    0     0     0 numeric   1213
7     ACCR       0    0.00    0    0     0     0 numeric   1239
8     LEVI       1    0.08    0    0     0     0 numeric   1238
9   target    1200   96.85    0    0     0     0  factor      2
```

   The data is clean with no null values. The proportion of non-manipulators in the target is 96.85%: Indicating highly unbalanced data

   Univariate:

```
summary(fulldata)
```

```
> summary(fulldata)
      DSRI              GMI                AQI                SGI                DEPI        
 Min.   : 0.0000   Min.   :-20.8118   Min.   :-32.8856   Min.   : 0.02769   Min.   :0.06882  
 1st Qu.: 0.8908   1st Qu.:  0.9271   1st Qu.:  0.7712   1st Qu.: 0.97021   1st Qu.:0.93690  
 Median : 1.0227   Median :  1.0000   Median :  1.0040   Median : 1.08896   Median :1.00191  
 Mean   : 1.1691   Mean   :  0.9879   Mean   :  0.9978   Mean   : 1.12709   Mean   :1.04014  
 3rd Qu.: 1.1925   3rd Qu.:  1.0580   3rd Qu.:  1.2163   3rd Qu.: 1.19998   3rd Qu.:1.08136  
 Max.   :36.2912   Max.   : 46.4667   Max.   : 52.8867   Max.   :13.08143   Max.   :5.39387  
      SGAI              ACCR               LEVI         target      
 Min.   : 0.0000   Min.   :-3.14350   Min.   : 0.0000   0:1200  
 1st Qu.: 0.8988   1st Qu.:-0.07633   1st Qu.: 0.9232   1:  39  
 Median : 1.0000   Median :-0.02924   Median : 1.0131           
 Mean   : 1.1072   Mean   :-0.03242   Mean   : 1.0571           
 3rd Qu.: 1.1300   3rd Qu.: 0.02252   3rd Qu.: 1.1156           
 Max.   :49.3018   Max.   : 0.95989   Max.   :13.0586           
> 
```

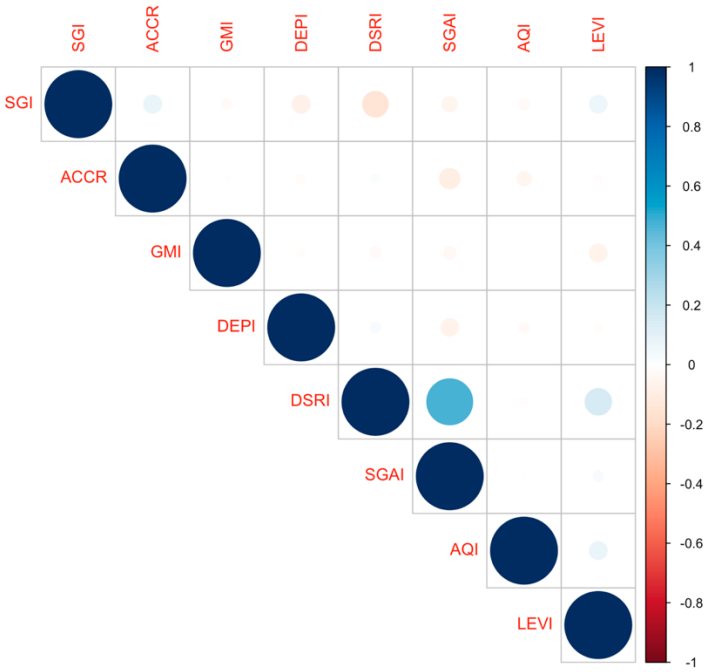
Outlier detection and behavior in each independent variable:

```
hist(fulldata$DSRI)              #Outliers exists after DSRI>10
f <- which(fulldata$DSRI>10)
f1 <- fulldata[f,]
f1                               #(3) observations with DSRI >10, all manipulators

hist(fulldata$GMI)               #Outliers exists at GMI>5 and GMI <-10
g <- which(fulldata$GMI>5 | fulldata$GMI< -10)
g1 <- fulldata[g,]        #Extreme positive high values of GMI has manipulator record (1)

hist(fulldata$AQI)               #Outliers exists at AQI>20 and AQI < -20
a <- which(fulldata$AQI>20 | fulldata$AQI< -20)
a1 <- fulldata[a,]               #Extreme positive high value has manipulator record (1)

hist(fulldata$SGI)               #Outliers exists after SGI>5
s <- which(fulldata$SGI>5)
s1 <- fulldata[s,]               #Extreme positive high values are manipulator records (2)

hist(fulldata$SGAI)              #Outliers exists after SGAI>10
sg <- which(fulldata$SGAI>10)
sg1 <- fulldata[sg,]             #Extreme high values are manipulator records (2)

hist(fulldata$LEVI)              #Outliers exists after LEVI>5
l <- which(fulldata$LEVI>5)
l1 <- fulldata[l,]               #Extreme high values are manipulator records (2/3)
```

From outlier detection, we can conclude that the outliers are important in the data and majority of these values constitute the proportion of manipulator class. Hence, outliers should not be removed.

Bivariate Analysis using Correlation Matrix:

```
fulldata_c <- subset(fulldata, select = -c(target))
res <- cor(fulldata_c)
round(res, 2)
library(corrplot)
corrplot(res, method="circle", type="upper", order = "hclust")
```



The plot clearly indicates that all numeric variables are not correlated.
Checking further for only DSRI and SGAI:
`cor(fulldata$DSRI, fulldata$SGAI)`
#0.470764: Not a high value; hence the data is fine to work with

a. Beneish model:
   Beneish M-Score = -4.84 + 0.92*DSRI + 0.528*GMI + 0.404*AQI + 0.892*SGI + 0.115*DEPI – 0.172*SGAI + 4.679*TATA – 0.327*LVGI
   For M-Score > -2.2: Organization is classified as a manipulator and vice versa.
   Source: https://www.gmtresearch.com/en/accounting-ratio/beneishs-m-score/

Pros:

- The model proposes appropriate variables to compute M-Score
- It is a renowned statistical model and hence direct computation of scores is faster and efficient.

Cons:

- Due to taxation financial policies being different in countries: US model is not valid in India. (Asian companies do not distinguish between COGS and SG&A.)
- Can be made valid for companies with a market capitalization exceeding US$1bn according to recent research but no such data to support that existence.

Practically:

```
m_score <- (-4.84) + (0.92*fulldata$DSRI) + (0.528*fulldata$GMI) + (0.404*fulldata$AQI) + (0.892*fulldata$SGI) + (0.115*fulldata$DEPI) - (0.172*fulldata$SGAI) + (4.679*fulldata$ACCR) - (0.327*fulldata$LEVI)
```

```
table(m_score> -2.22)
#FALSE  TRUE
#829   410
```

```
library(caret)
confusionMatrix(m_score, fulldata$target, positive = '1')
```

```
          Reference
Prediction   0   1
         0 829   0
         1 371  39

               Accuracy : 0.7006
                 95% CI : (0.6742, 0.726)
    No Information Rate : 0.9685
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1233

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 1.00000
            Specificity : 0.69083
         Pos Pred Value : 0.09512
         Neg Pred Value : 1.00000
             Prevalence : 0.03148
         Detection Rate : 0.03148
   Detection Prevalence : 0.33091
      Balanced Accuracy : 0.84542

       'Positive' Class : 1
```

Even though: True Positive= 39; False Positive = 371
Even though, the model correctly identifies all the True Positives; it doesn't do the same for True Negatives. 371/ 1200 were incorrectly classified. And since, even minimizing false positive

is important and should not be compromised at the cost of false negative minimization. Hence, we require a machine learning model to adjust this trade off.

b. Problems expected when cases in one class are much lower than the other (in binary classification): Due to highly unbalanced data, the model gets trained to almost always classify the observation as majority class label. Since, models work on accuracy and minimizing error rate: this leads to compromising on identifying correct classification of minority class.

   Models robust to unbalanced data:
   Ensemble methods may learn over the period of time about the minority class due to misclassified points in first few attempts. And hence, a high number of iterations in ensemble methods might be a good option to deal with. For example: ADA Boost, Bagging and Random Forest.

   Techniques that can be adopted for unbalanced data:
   - Undersampling: Decreasing the size of majority class to the size of minority class
   - Oversampling: Repeating minority class observations to match the size of majority class
   - SMOTE: Synthetically creating minority class observations based on their values, behavior and proximity to match the size of majority class.

c. Using sampled data:
   Reading and treating the data: (The same as done for full data)

```
library(readxl)
sample_data <- read_excel("Desktop/IMB579-XLS-ENG.xlsx", sheet = "Sample for Model Development")
View(sample_data)
sampledata <- subset(sample_data, select = -c(`Company ID`, Manipulator))
rm(sample_data)
View(sampledata)
sampledata$`C-MANIPULATOR` <- as.factor(sampledata$`C-MANIPULATOR`)
sampledata <- sampledata %>% rename(target = `C-MANIPULATOR` )
```

   Under sampling:
   Cons of the method:
   - Losing on important information
   - Samples of majority class may be very different

   Splitting data:

```
set.seed(12345)
index <- sample(2, nrow(sampledata), replace = T, prob = c(0.7,0.3))
```

```
TrainData <- sampledata[index == 1, ]
TestData <- sampledata[index == 2, ]
```

Checking proportion of majority and minority class in Train and Test:

```
prop.table(table(TrainData$target))
prop.table(table(TestData$target))
```

```
> prop.table(table(TrainData$target))

        0         1
0.8156028 0.1843972
> prop.table(table(TestData$target))

       0        1
0.835443 0.164557
>
```

Since the proportion in almost the same, we can under sample the train data.

```
library(ROSE)
under <- ovun.sample(target~., data=TrainData,
         p=0.5, seed=123,
         method="under")$data
```

Under sampling the train data to attain target variable's equal proportion for minority and majority class. The seed is set at 123 to attain 1 sample, and use that throughout.

```
table(under$target)
```

0: 25, 1: 26

The proportion of minority and majority is same now.

Stepwise Logistic Regression:

```
full <- glm(target~., data= under, family= "binomial")
null <- glm(target~1, data= under, family= "binomial")
stepf <- step(null, scope= list(lower= null, upper= full), direction = "both")
summary(stepf)
```

```
> summary(stepf)

Call:
glm(formula = target ~ ACCR + AQI + DSRI + SGI + LEVI, family = "binomial",
    data = under)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.8160  -0.3429   0.0000   0.0988   1.7820

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -15.258      5.801  -2.630  0.00854 **
ACCR          22.488      9.974   2.255  0.02416 *
AQI            2.003      0.934   2.144  0.03201 *
DSRI           6.519      3.103   2.101  0.03563 *
SGI            6.542      2.648   2.471  0.01348 *
LEVI          -2.447      1.393  -1.757  0.07886 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 70.681  on 50  degrees of freedom
Residual deviance: 21.825  on 45  degrees of freedom
AIC: 33.825

Number of Fisher Scoring iterations: 11
```

Variable selection leading to model selection :
glm(formula = target ~ ACCR + AQI + DSRI + SGI + LEVI, family = "binomial",
data = under)
AIC value reduced from: 72.68 to 33.825

Multiple samples cross-check:
Different sample output:

```
Call:
glm(formula = target ~ ACCR + AQI + DSRI + SGI + DEPI + LEVI
 +
    SGAI, family = "binomial", data = under1)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.86921  -0.04345  -0.00023   0.03902   1.65245

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -66.269     29.871  -2.218   0.0265 *
ACCR          22.179      9.235   2.402   0.0163 *
AQI            2.793      1.324   2.109   0.0350 *
DSRI          14.197      6.495   2.186   0.0288 *
SGI           22.633     10.298   2.198   0.0280 *
DEPI          31.084     15.101   2.058   0.0395 *
LEVI          -6.568      3.093  -2.123   0.0337 *
SGAI          -5.535      3.270  -1.693   0.0905 .
---
Signif. codes:
0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 76.082  on 54  degrees of freedom
Residual deviance: 12.758  on 47  degrees of freedom
AIC: 28.758

Number of Fisher Scoring iterations: 11
```

4 out of the 5 variables were constant in 7 samples that were checked: hence continuing with the model created first.

Probability equations:

P/1-P = e^(-15.258 + 22.488*ACCR. + 2.003*AQI + 6.519* DSRI + 6.542*SGI -2.447*LEVI)

P (class =1)
= e^(-15.258 + 22.488*ACCR. + 2.003*AQI + 6.519* DSRI + 6.542*SGI -2.447*LEVI) /
{ 1+ e^(-15.258 + 22.488*ACCR. + 2.003*AQI + 6.519* DSRI + 6.542*SGI -2.447*LEVI}

P (class = 0)
= 1 /
{ 1+ e^(-15.258 + 22.488*ACCR. + 2.003*AQI + 6.519* DSRI + 6.542*SGI - 2.447*LEVI}

d. Evaluating the model:

Predicting:

```
Pred <- predict(stepf, newdata= TestData, type="response")
Pred
Class <- ifelse(Pred >= 0.5, '1', "0")
Class <- as.factor(Class)
```

```
> confusionMatrix(Class, TestData$target, positive = '1')
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 46  1
         1 20 12

               Accuracy : 0.7342
                 95% CI : (0.6228, 0.8273)
    No Information Rate : 0.8354
    P-Value [Acc > NIR] : 0.9925

                  Kappa : 0.3907

 Mcnemar's Test P-Value : 0.00008568

            Sensitivity : 0.9231
            Specificity : 0.6970
         Pos Pred Value : 0.3750
         Neg Pred Value : 0.9787
             Prevalence : 0.1646
         Detection Rate : 0.1519
   Detection Prevalence : 0.4051
      Balanced Accuracy : 0.8100

       'Positive' Class : 1
```

#Using F-Score for different cut-off and improving the result
Using F-Score because: Precision and Recall both are important in this situation. Using 'beta' value to adjust weights for each class in F-Score.
The formula of F-Score is
(1+beta^2) Precision * Recall / (beta^2) Precision +Recall
Since Precision minimizes False Negative, and taking positive class as '1': Precision should be weighted more than Recall.

Assigning 60-40 weights to Precision and Recall respectively: (B: Beta)
B^2 / (1+ B^2) = 60/100
Solving the equation:
2 * (B ^2) = 3
(B ^ 2) = 3/2

Code to find optimal cut off point in accordance with F-Score:

```
s <- seq(from = 0, to= 1, by = 0.1)
fun = function(s)
 {
 Class <- ifelse(Pred >= s, '1', '0')
 Class <- as.factor(Class)
 c_recall <- confusionMatrix(Class, TestData$target, positive = '1')$byClass['Recall']
 c_precision <- confusionMatrix(Class, TestData$target, positive = '1')$byClass['Precision']
 f_score <- (2.5*c_recall*c_precision)/ (1.5*c_precision + c_recall)
 return(f_score)
 }

which.max(lapply(s, fun))  #9
s[9]                       #Optimal Cut off: 0.8
```

Predicting with new cut-off:

```
Pred <- predict(stepf, newdata= TestData, type="response")
Class <- ifelse(Pred >= 0.8, '1', "0")
Class <- as.factor(Class)
confusionMatrix(Class, TestData$target, positive = '1')
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 55  3
         1 11 10

               Accuracy : 0.8228
                 95% CI : (0.7206, 0.8996)
    No Information Rate : 0.8354
    P-Value [Acc > NIR] : 0.68501

                  Kappa : 0.4832

 Mcnemar's Test P-Value : 0.06137

            Sensitivity : 0.7692
            Specificity : 0.8333
         Pos Pred Value : 0.4762
         Neg Pred Value : 0.9483
             Prevalence : 0.1646
         Detection Rate : 0.1266
   Detection Prevalence : 0.2658
      Balanced Accuracy : 0.8013

       'Positive' Class : 1
```

Results have improved.

Cut-off probability as 0.5:

As observed, a cut-off probability of 0.5 is not feasible for this study as that leads to average results. Hence, F-Score was obtained to identify optimal cut-off point.

Different cut-off points and results:

```
s <- seq(from = 0, to= 1, by = 0.1)
fun = function(s)
{
 Class <- ifelse(Pred >= s, '1', '0')
 Class <- as.factor(Class)
 return(confusionMatrix(Class, TestData$target, positive = '1')$byClass)
}

lapply(s, fun)
```

For cut-off points: 0 through 1, incremented by 0.1:

```
> lapply(s, fun)
[[1]]
         Sensitivity          Specificity       Pos Pred Value       Neg Pred Value
           1.0000000            0.0000000            0.1645570                  NaN
           Precision               Recall                   F1           Prevalence
           0.1645570            1.0000000            0.2826087            0.1645570
      Detection Rate Detection Prevalence    Balanced Accuracy
           0.1645570            1.0000000            0.5000000

[[2]]
         Sensitivity          Specificity       Pos Pred Value       Neg Pred Value
           1.0000000            0.5000000            0.2826087            1.0000000
           Precision               Recall                   F1           Prevalence
           0.2826087            1.0000000            0.4406780            0.1645570
      Detection Rate Detection Prevalence    Balanced Accuracy
           0.1645570            0.5822785            0.7500000

[[3]]
         Sensitivity          Specificity       Pos Pred Value       Neg Pred Value
           0.9230769            0.5757576            0.3000000            0.9743590
           Precision               Recall                   F1           Prevalence
           0.3000000            0.9230769            0.4528302            0.1645570
      Detection Rate Detection Prevalence    Balanced Accuracy
           0.1518987            0.5063291            0.7494172

[[4]]
         Sensitivity          Specificity       Pos Pred Value       Neg Pred Value
           0.9230769            0.6212121            0.3243243            0.9761905
           Precision               Recall                   F1           Prevalence
           0.3243243            0.9230769            0.4800000            0.1645570
      Detection Rate Detection Prevalence    Balanced Accuracy
           0.1518987            0.4683544            0.7721445
```

```
[[5]]
        Sensitivity          Specificity       Pos Pred Value       Neg Pred Value
          0.9230769            0.6666667            0.3529412            0.9777778
          Precision               Recall                   F1           Prevalence
          0.3529412            0.9230769            0.5106383            0.1645570
     Detection Rate Detection Prevalence    Balanced Accuracy
          0.1518987            0.4303797            0.7948718

[[6]]
        Sensitivity          Specificity       Pos Pred Value       Neg Pred Value
          0.9230769            0.6969697            0.3750000            0.9787234
          Precision               Recall                   F1           Prevalence
          0.3750000            0.9230769            0.5333333            0.1645570
     Detection Rate Detection Prevalence    Balanced Accuracy
          0.1518987            0.4050633            0.8100233

[[7]]
        Sensitivity          Specificity       Pos Pred Value       Neg Pred Value
          0.9230769            0.7272727            0.4000000            0.9795918
          Precision               Recall                   F1           Prevalence
          0.4000000            0.9230769            0.5581395            0.1645570
     Detection Rate Detection Prevalence    Balanced Accuracy
          0.1518987            0.3797468            0.8251748

[[8]]
        Sensitivity          Specificity       Pos Pred Value       Neg Pred Value
          0.9230769            0.7272727            0.4000000            0.9795918
          Precision               Recall                   F1           Prevalence
          0.4000000            0.9230769            0.5581395            0.1645570
     Detection Rate Detection Prevalence    Balanced Accuracy
          0.1518987            0.3797468            0.8251748


[[9]]
        Sensitivity          Specificity       Pos Pred Value       Neg Pred Value
          0.7692308            0.8333333            0.4761905            0.9482759
          Precision               Recall                   F1           Prevalence
          0.4761905            0.7692308            0.5882353            0.1645570
     Detection Rate Detection Prevalence    Balanced Accuracy
          0.1265823            0.2658228            0.8012821

[[10]]
        Sensitivity          Specificity       Pos Pred Value       Neg Pred Value
          0.7692308            0.8333333            0.4761905            0.9482759
          Precision               Recall                   F1           Prevalence
          0.4761905            0.7692308            0.5882353            0.1645570
     Detection Rate Detection Prevalence    Balanced Accuracy
          0.1265823            0.2658228            0.8012821

[[11]]
        Sensitivity          Specificity       Pos Pred Value       Neg Pred Value
           0.000000            1.000000                  NaN             0.835443
          Precision               Recall                   F1           Prevalence
                 NA            0.000000                   NA            0.164557
     Detection Rate Detection Prevalence    Balanced Accuracy
           0.000000            0.000000             0.500000
```

Trend:

Sensitivity decreases from cut-off at 0 to 1 and specificity increases from cut-off at 0 to 1. Optimal values of both of them can be seen at [[9]] and [[10]], hence the cut-off chosen was 0.8.

e. Best cut-off point using indexes given:

i. Youden's Index:

Given the parameters: Created a function which calculates sensitivity and specificity from the confusion matrix on a sequence of numbers used as cut-off probabilities. The sensitivity and specificity are used to calculate Youden's Index on loop. The optimal value is where Youden's index in maximum, hence which.max(output of the function) is used.

```
s <- seq(from = 0, to= 1, by = 0.001)

fun = function(s)
{
 Class <- ifelse(Pred >= s, '1', "0")
 Class <- as.factor(Class)
 i <- confusionMatrix(Class, TestData$target, positive = '1')$byClass['Sensitivity']
 j <- confusionMatrix(Class, TestData$target, positive = '1')$byClass['Specificity']
 return(i+j-1)
}

which.max(lapply(s, fun))      #776
s[776]                         #0.775
```

<u>Predicting:</u>

```
Pred <- predict(stepf, newdata= TestData, type="response")
Class <- ifelse(Pred >= 0.775, '1', "0")
Class <- as.factor(Class)
confusionMatrix(Class, TestData$target, positive = '1')
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 52  3
         1 14 10

               Accuracy : 0.7848
                 95% CI : (0.678, 0.8694)
    No Information Rate : 0.8354
    P-Value [Acc > NIR] : 0.91023

                  Kappa : 0.4158

 Mcnemar's Test P-Value : 0.01529

            Sensitivity : 0.7692
            Specificity : 0.7879
         Pos Pred Value : 0.4167
         Neg Pred Value : 0.9455
             Prevalence : 0.1646
         Detection Rate : 0.1266
   Detection Prevalence : 0.3038
      Balanced Accuracy : 0.7786

       'Positive' Class : 1
```

Implementation: Slightly better results than F-Score

Cost- based method

Created a function by calculation false negative and false positive from table's indexes and storing it respectively. These values were multiplied with the penalties and the function was run on a sequence of 0 to 1, incremented by 0.001 to find minimum value of returned cost.
Even though minimizing False Negative and False Positive both are important; assigning more penalty to False negative than False Positive considering incorrect classification of manipulators should be minimized over incorrect classification of non-manipulators.

```
p1=7
p2=3
s <- seq(from = 0, to= 1, by = 0.001)
fun = function(s)
{
 Class <- ifelse(Pred >= s, '1', "0")
 Class <- as.factor(Class)
 #p10
 false_negative <- confusionMatrix(Class, TestData$target, positive = '1')$table[1,2]
```

```
 #p01
 false_positive <- confusionMatrix(Class, TestData$target, positive = '1')$table[2,1]
 return((p1*false_negative + p2*false_positive))     #Cost
}

which.min(lapply(s, fun))       #951
s[951]          #0.95
```

Predicting using Cost based method:

```
Class <- ifelse(Pred >= 0.95, '1', "0")
Class <- as.factor(Class)
confusionMatrix(Class, TestData$target, positive = '1')
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 57  3
         1  9 10

               Accuracy : 0.8481
                 95% CI : (0.7497, 0.919)
    No Information Rate : 0.8354
    P-Value [Acc > NIR] : 0.4531

                  Kappa : 0.5339

 Mcnemar's Test P-Value : 0.1489

            Sensitivity : 0.7692
            Specificity : 0.8636
         Pos Pred Value : 0.5263
         Neg Pred Value : 0.9500
             Prevalence : 0.1646
         Detection Rate : 0.1266
   Detection Prevalence : 0.2405
      Balanced Accuracy : 0.8164

       'Positive' Class : 1
```

So far, the best trade off can be seen here.

f. Considering the best model so far as follows:

```
full1 <- glm(target~., data= under1, family= "binomial")
null1 <- glm(target~1, data= under1, family= "binomial")
stepf1 <- step(null1, scope= list(lower= null1, upper= full1), direction = "both")
summary(stepf1)

Pred <- predict(stepf, newdata= TestData, type="response")
Class <- ifelse(Pred >= 0.95, '1', "0")
Class <- as.factor(Class)
confusionMatrix(Class, TestData$target, positive = '1')
```

Defining M-Score for this model:

For 'n' number of independent variables:
{x = b0 + b1*x1 + b2*x2 + … + bn*xn}

For optimal cut off point as 0.95:
e^(x) / {1+ e^(x)} = P
e^(x) / {1+ e^(x)} = 0.95

Solving the equation:
x = ln {0.95/(1-0.95)}
x = ln 19
x = 2.944

Therefore:
If: b0 + b1*x1 + b2*x2 + … + bn*xn >= 2.944; target class = 1
and if b0 + b1*x1 + b2*x2 + … + bn*xn < 2.944; target class = 0

Equation for M-Score:
-15.258 + 22.488*ACCR. + 2.003*AQI + 6.519* DSRI + 6.542*SGI -2.447*LEVI > 2.944 : classified as manipulator

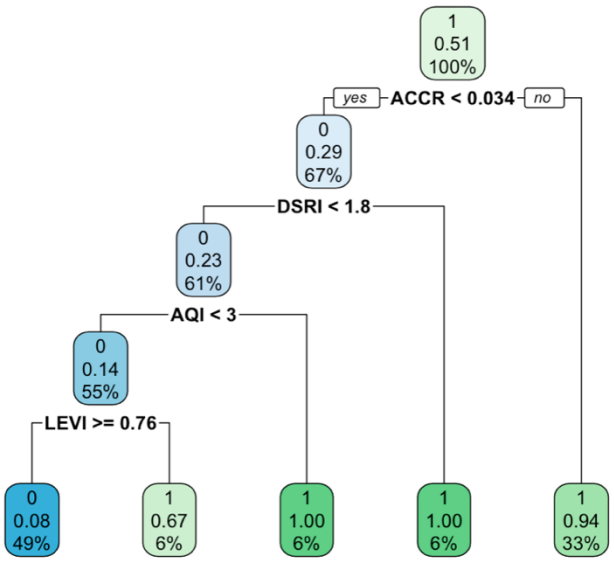
g. CART:

- with undersampled (balanced data):

```
library(rpart)
library(rpart.plot)
r <- rpart(target~., data= under,control = rpart.control(minsplit =10, cp = 0.01))
r
```

```
> r
n= 51

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

 1) root 51 25 1 (0.49019608 0.50980392)
   2) ACCR< 0.03426652 34 10 0 (0.70588235 0.29411765)
     4) DSRI< 1.837973 31  7 0 (0.77419355 0.22580645)
       8) AQI< 3.029928 28  4 0 (0.85714286 0.14285714)
        16) LEVI>=0.7640077 25  2 0 (0.92000000 0.08000000) *
        17) LEVI< 0.7640077 3  1 1 (0.33333333 0.66666667) *
       9) AQI>=3.029928 3  0 1 (0.00000000 1.00000000) *
     5) DSRI>=1.837973 3  0 1 (0.00000000 1.00000000) *
   3) ACCR>=0.03426652 17  1 1 (0.05882353 0.94117647) *
```

```
rpart.plot(r)
```



Key predictors: ACCR, DSRI, AQI, LEVI

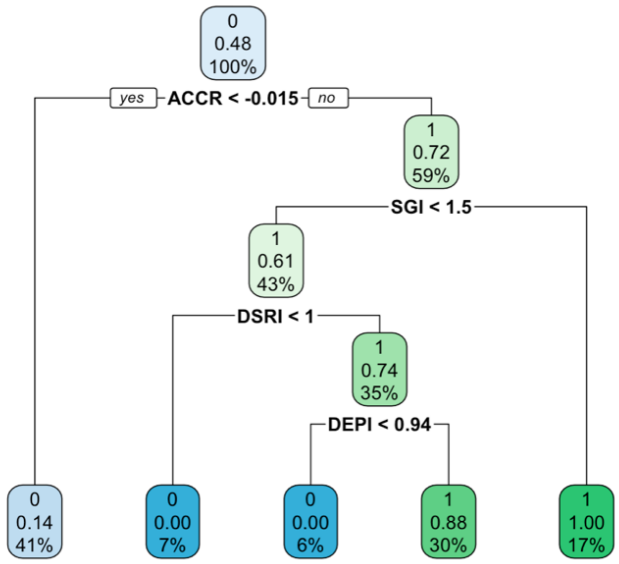
Since, decision tree is prone to variance, under sampling might create very different results for different samples. Checking before decising best rules:

- Second under sample:

```
library(ROSE)
under1 <- ovun.sample(target~., data=TrainData,
                      p=0.5, seed=1,
                      method="under")$data
r1 <- rpart(target~., data= under1, control = rpart.control(minsplit =10, cp = 0.01))
r1
```

```
> r1
n= 54

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

 1) root 54 26 0 (0.5185185 0.4814815)
   2) ACCR< -0.01475807 22  3 0 (0.8636364 0.1363636) *
   3) ACCR>=-0.01475807 32  9 1 (0.2812500 0.7187500)
     6) SGI< 1.459574 23  9 1 (0.3913043 0.6086957)
      12) DSRI< 0.9989312 4  0 0 (1.0000000 0.0000000) *
      13) DSRI>=0.9989312 19  5 1 (0.2631579 0.7368421)
        26) DEPI< 0.9401263 3  0 0 (1.0000000 0.0000000) *
        27) DEPI>=0.9401263 16  2 1 (0.1250000 0.8750000) *
     7) SGI>=1.459574 9  0 1 (0.0000000 1.0000000) *
> rpart.plot(r1)
```

```
rpart.plot(r1)
```



Key predictors: ACCR, SGI, DSRI, DEPI

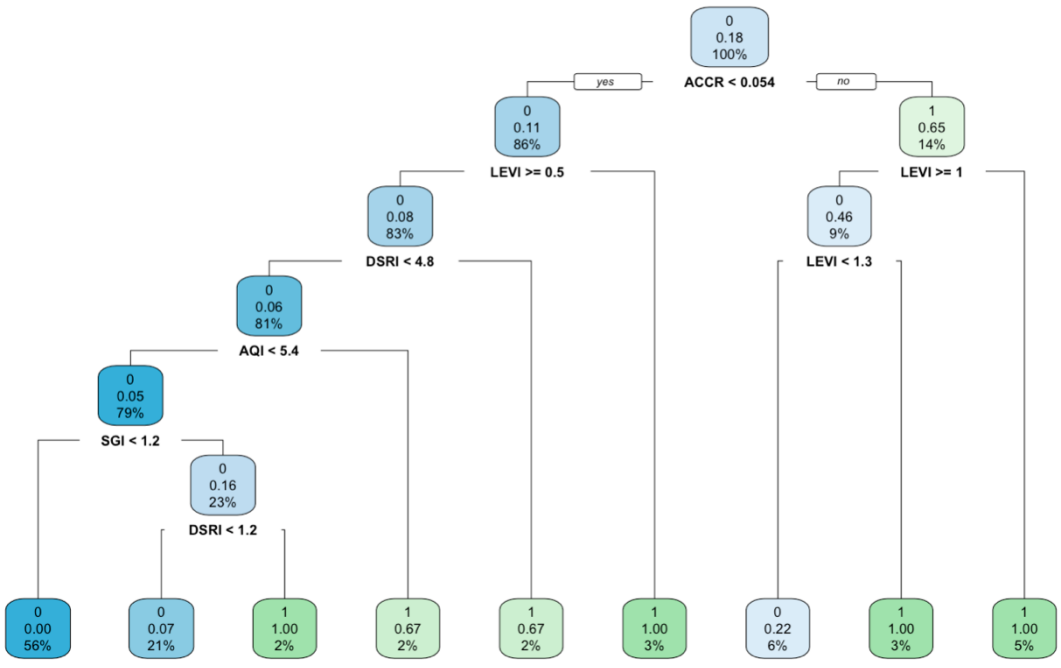
Since, key predictors are changing with change in sample, considering full data's train data to create a decision tree and device rules.

- With Full Data

```
r2 <- rpart(target~., data= TrainData, control = rpart.control(minsplit =10, cp = 0.01))
r2
```

```
> r2
n= 141

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

 1) root 141 26 0 (0.81560284 0.18439716)
   2) ACCR< 0.05388837 121 13 0 (0.89256198 0.10743802)
     4) LEVI>=0.4978521 117  9 0 (0.92307692 0.07692308)
       8) DSRI< 4.829657 114  7 0 (0.93859649 0.06140351)
        16) AQI< 5.436298 111  5 0 (0.95495495 0.04504505)
          32) SGI< 1.197355 79  0 0 (1.00000000 0.00000000) *
          33) SGI>=1.197355 32  5 0 (0.84375000 0.15625000)
            66) DSRI< 1.225592 29  2 0 (0.93103448 0.06896552) *
            67) DSRI>=1.225592 3  0 1 (0.00000000 1.00000000) *
        17) AQI>=5.436298 3  1 1 (0.33333333 0.66666667) *
       9) DSRI>=4.829657 3  1 1 (0.33333333 0.66666667) *
     5) LEVI< 0.4978521 4  0 1 (0.00000000 1.00000000) *
   3) ACCR>=0.05388837 20  7 1 (0.35000000 0.65000000)
     6) LEVI>=1.007158 13  6 0 (0.53846154 0.46153846)
      12) LEVI< 1.303391 9  2 0 (0.77777778 0.22222222) *
      13) LEVI>=1.303391 4  0 1 (0.00000000 1.00000000) *
     7) LEVI< 1.007158 7  0 1 (0.00000000 1.00000000) *
```

```
rpart.plot(r2)
```



Best Decision Rules:

Out of the rules obtained for each class, selecting the rules with highest confidence and support in priority order:

- For class 0: Rule 32 has the highest support: 56% and loss = 0%

Rule: IF (ACCR < 0.054) & (LEVI >= 0.5) & (DSRI < 4.8) & (AQI < 5.4) & (SGI < 1.197), Then Class = 0, else Class = 1

- For class 0: Rule 66 has the second highest support: 21% and loss = 0%

Rule: IF (ACCR < 0.054) & (LEVI >= 0.5) & (DSRI < 4.8) & (AQI < 5.4) & (SGI > 1.197) & (DSRI < 1.2),
Then Class = 0, else Class = 1

- For class 1: Rule 7 has the highest support: 5% and confidence = 100%

Rule: IF (ACCR > 0.0554) & (LEVI >= 1), Then Class 1, else Class 0

- For unbalanced data, support for class 1 would be less and hence, rules 13 and 5 can also be considered as good rules with 3% support and 100% confidence.

Predicting on the tree formed:

```
p <- predict(r2, newdata = TestData, type = "class")
p <- as.factor(p)
confusionMatrix(p, TestData$target, positive = '1')
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 58  4
         1  8  9

               Accuracy : 0.8481
                 95% CI : (0.7497, 0.919)
    No Information Rate : 0.8354
    P-Value [Acc > NIR] : 0.4531

                  Kappa : 0.5083

 Mcnemar's Test P-Value : 0.3865

            Sensitivity : 0.6923
            Specificity : 0.8788
         Pos Pred Value : 0.5294
         Neg Pred Value : 0.9355
             Prevalence : 0.1646
         Detection Rate : 0.1139
   Detection Prevalence : 0.2152
      Balanced Accuracy : 0.7855

       'Positive' Class : 1
```

#A really good model with comparison to logistic- cost based cut-off.

h. Logistic regression model with complete data:

Data Split:

```
str(fulldata)
index <- sample(2, nrow(fulldata), replace = T, prob = c(0.7,0.3))
TrainData <- fulldata[index == 1, ]
TestData <- fulldata[index == 2, ]

prop.table(table(TrainData$target))
#0                1
#0.96937574  0.03062426
prop.table(table(TestData$target))
#0                1
#0.96666667  0.03333333
```

Stepwise Logistic Regression Model (to do a better feature selection):

```
full <- glm(target~., data= TrainData, family= "binomial")
null <- glm(target~1, data= TrainData, family= "binomial")
stepf <- step(null, scope= list(lower= null, upper= full), direction = "both")
summary(stepf)
```

```
> summary(stepf)

Call:
glm(formula = target ~ DSRI + SGI + ACCR + AQI + SGAI + LEVI,
    family = "binomial", data = TrainData)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.8247  -0.1626  -0.1139  -0.0850   3.2280

Coefficients:
            Estimate Std. Error z value             Pr(>|z|)
(Intercept) -7.99225    0.89717  -8.908 < 0.0000000000000002 ***
DSRI         0.65104    0.15704   4.146          0.000033894 ***
SGI          2.16309    0.48567   4.454          0.000008436 ***
ACCR         9.39609    1.92080   4.892          0.000000999 ***
AQI          0.27388    0.09498   2.884              0.00393 **
SGAI         0.60177    0.32988   1.824              0.06812 .
LEVI        -0.58428    0.32550  -1.795              0.07265 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 211.64  on 853  degrees of freedom
Residual deviance: 124.69  on 847  degrees of freedom
AIC: 138.69

Number of Fisher Scoring iterations: 8
```

```
Pred <- predict(stepf, newdata= TestData, type = "response")
```

```
Pred
Class <- ifelse(Pred >= 0.5, '1', "0")
Class <- as.factor(Class)
confusionMatrix(Class, TestData$target, positive = '1')
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 367  13
         1   2   3

               Accuracy : 0.961
                 95% CI : (0.9366, 0.978)
    No Information Rate : 0.9584
    P-Value [Acc > NIR] : 0.464633

                  Kappa : 0.2713

 Mcnemar's Test P-Value : 0.009823

            Sensitivity : 0.187500
            Specificity : 0.994580
         Pos Pred Value : 0.600000
         Neg Pred Value : 0.965789
             Prevalence : 0.041558
         Detection Rate : 0.007792
   Detection Prevalence : 0.012987
      Balanced Accuracy : 0.591040

       'Positive' Class : 1
```

A very bad model for predicting manipulators because of high false negatives.

Finding optimal cut-off point using cost based method to improve the model:

```
p1=7
p2=3
s <- seq(from = 0, to= 1, by = 0.01)
fun = function(s)
{
 Class <- ifelse(Pred >= s, '1', "0")
 Class <- as.factor(Class)
 #p10
 false_negative <- confusionMatrix(Class, TestData$target, positive = '1')$table[1,2]
 #p01
 false_positive <- confusionMatrix(Class, TestData$target, positive = '1')$table[2,1]
 return((p1*false_negative + p2*false_positive))
}
```

```
which.min(lapply(s, fun))
s[47] #0.46
```

Predicting using the new cut-off:

```
Pred <- predict(stepf, newdata= TestData, type = "response")
Pred
Class <- ifelse(Pred >= 0.46, '1', "0")
Class <- as.factor(Class)
confusionMatrix(Class, TestData$target, positive = '1')
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 367  13
         1   2   3

               Accuracy : 0.961
                 95% CI : (0.9366, 0.978)
    No Information Rate : 0.9584
    P-Value [Acc > NIR] : 0.464633

                  Kappa : 0.2713

 Mcnemar's Test P-Value : 0.009823

            Sensitivity : 0.187500
            Specificity : 0.994580
         Pos Pred Value : 0.600000
         Neg Pred Value : 0.965789
             Prevalence : 0.041558
         Detection Rate : 0.007792
   Detection Prevalence : 0.012987
      Balanced Accuracy : 0.591040

       'Positive' Class : 1
```

Similar bad results.

Predicting on test data of complete data using the model trained on sample data's under sampled trained data:

```
full1 <- glm(target~., data= under1, family= "binomial")
null1 <- glm(target~1, data= under1, family= "binomial")
stepf1 <- step(null1, scope= list(lower= null1, upper= full1), direction = "both")

Pred <- predict(stepf1, newdata= TestData, type="response")
Class <- ifelse(Pred >= 0.95, '1', "0")
Class <- as.factor(Class)
confusionMatrix(Class, TestData$target, positive = '1')
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 335   6
         1  34  10

               Accuracy : 0.8961
                 95% CI : (0.8612, 0.9247)
    No Information Rate : 0.9584
    P-Value [Acc > NIR] : 1

                  Kappa : 0.2901

 Mcnemar's Test P-Value : 0.00001963

            Sensitivity : 0.62500
            Specificity : 0.90786
         Pos Pred Value : 0.22727
         Neg Pred Value : 0.98240
             Prevalence : 0.04156
         Detection Rate : 0.02597
   Detection Prevalence : 0.11429
      Balanced Accuracy : 0.76643

       'Positive' Class : 1
```

#Much better results for classifying manipulators

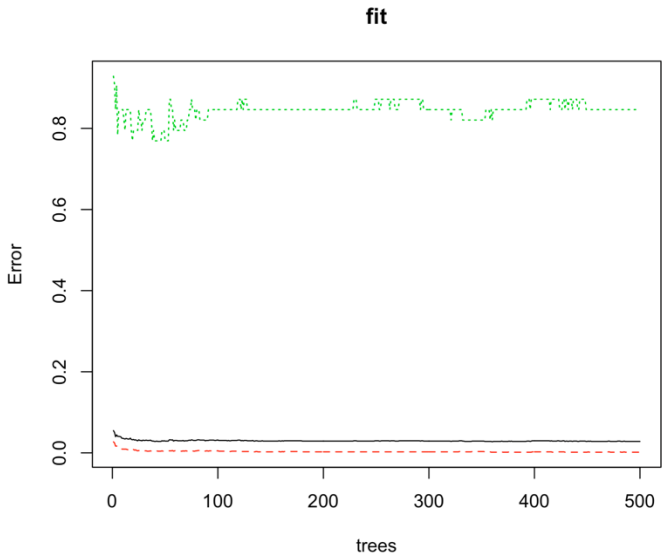
i. Random Forest:

```
library(randomForest)
fit = randomForest(target~., data=fulldata, ntree=500,
            importance=TRUE, proximity=TRUE)
#OOB estimate of  error rate: 2.91%
#Confusion matrix:
#     0 1 class.error
#0 1197 3   0.0025000
#1   33 6   0.8461538
```

```
importance(fit)
```

```
> importance(fit)
                0          1 MeanDecreaseAccuracy MeanDecreaseGini
DSRI 14.79421506 13.5526456           17.4111982        12.230369
GMI  -0.46204657 -0.2507031           -0.5098718         5.440007
AQI  -0.03679904  1.9160806            0.7155560         6.950588
SGI  10.81089268 10.9364886           13.1572609        12.066521
DEPI  0.92492447  0.9465548            1.1126419         6.111878
SGAI -1.27924529  0.3753872           -0.9363701         8.131068
ACCR -1.81446687 12.3619434            3.6613980        10.252830
LEVI 10.99680386 19.1683925           17.9843579        14.443424
```

```
plot(fit)
```



From the Graph, we can see that less trees will have a better error as compared to more tress:

```
library(randomForest)
```

```
fit = randomForest(target~., data=fulldata, ntree=8,
                   importance=TRUE, proximity=TRUE)
#OOB estimate of  error rate: 2.86%
#Confusion matrix:
#     0  1 class.error
#0 1178  7 0.005907173
#1   28 10 0.736842105
```

#Better classification than with random forest with more trees.

Comparing it with logistic regression:
Logistic regression model under sampled on sample data performed better than random forest.

ii. Ada Boost:

Splitting data:

```
index <- sample(2, nrow(fulldata), replace = T, prob = c(0.7,0.3))
TrainData <- fulldata[index == 1, ]
TestData <- fulldata[index == 2, ]
install.packages("adabag")
library("adabag")

under <- ovun.sample(target~., data=TrainData, p=0.5, seed=123,
method="under")$data
b <- boosting(target ~ ., data = under, mfinal = 100, control = rpart.control(maxdepth =
4))

b$class
b$class <- as.factor(b$class)

table(b$class, under$target, dnn = c("Predicted Class", "Observed Class"))
#                 Observed Class
# Predicted Class  0  1
#                 0 24  0
#                 1  0 24
```

100% accuracy on training data

```
pred <- predict.boosting(b, newdata = as.data.frame(TestData))
#            Observed Class
#Predicted Class   0   1
#                  0 278  5
#                  1  52  10
```

The accuracy is average, but true positive is high, the model could have been better with entire data.

j. Final Recommendation:
Ada boost on the entire data would yield best results as can be seen from the results of under sampled data.
Logistic regression under sampled on train data yielded by far the greatest results and would recommend that after ensemble methods. The cut-off used should be cost based in this case.
Decision tree on the entire data was better than the decision tree on under sampled data, and hence would recommend decision tree for forming decisions and understanding important key performing indicators.