‘PREDICTING THE RISK OF HEART DISEASE : A CLASSIFICATION PROBLEM WITH LOGISTIC REGRESSION, RANDOM FOREST, K NEAREST NEIGHBOURS(KNN) and NAIVE BAYES CLASSIFIER’

AGNIVA KONAR, UPASYA BOSE, TIYASHA SAMANTA

08/12/2021

The necessary packages are installed.

```
#importing the dataset from MS Excel and basic data manipulation

dataset=read_excel(file.choose())
View(dataset)
a=dataset[,-c(3,6,8,13,17,19,21)]
View(a)

str(a) #giving the structure of the dataset
```

```
## tibble [4,238 x 16] (S3: tbl_df/tbl/data.frame)
##  $ Gender         : num [1:4238] 1 0 1 0 0 0 0 0 1 1 ...
##  $ age            : num [1:4238] 39 46 48 61 46 43 63 45 52 43 ...
##  $ education (new): num [1:4238] 4 2 1 3 3 2 1 2 1 1 ...
##  $ currentSmoker  : num [1:4238] 0 0 1 1 1 0 0 1 0 1 ...
##  $ cigsPerDay(new): num [1:4238] 0 0 20 30 23 0 0 20 0 30 ...
##  $ BPMeds(new)    : num [1:4238] 0 0 0 0 0 0 0 0 0 0 ...
##  $ prevalentStroke: num [1:4238] 0 0 0 0 0 0 0 0 0 0 ...
##  $ prevalentHyp   : num [1:4238] 0 0 0 1 0 1 0 0 1 1 ...
##  $ diabetes       : num [1:4238] 0 0 0 0 0 0 0 0 0 0 ...
##  $ totChol(new)   : num [1:4238] 195 250 245 225 285 228 205 313 260 225 ...
##  $ sysBP          : num [1:4238] 106 121 128 150 130 ...
##  $ diaBP          : num [1:4238] 70 81 80 95 84 110 71 71 89 107 ...
##  $ BMI(new)       : num [1:4238] 27 28.7 25.3 28.6 23.1 ...
##  $ heartRate(new) : num [1:4238] 80 95 75 65 85 77 60 79 76 93 ...
##  $ glucose(new)   : num [1:4238] 77 76 70 103 85 99 85 78 79 88 ...
##  $ TenYearCHD     : num [1:4238] 0 0 0 1 0 0 1 0 0 0 ...
```

```
a$Gender=ifelse(test=a$Gender==0,yes="F",no="M")
a$Gender=as.factor(a$Gender)

a$currentSmoker=ifelse(test=a$currentSmoker==1,yes="Y",no="N")
a$currentSmoker=as.factor(a$currentSmoker)

a$`BPMeds(new)`=ifelse(test=a$`BPMeds(new)`==1,yes="Y",no="N")
a$`BPMeds(new)`=as.factor(a$`BPMeds(new)`)

a$prevalentStroke=ifelse(test=a$prevalentStroke==1,yes="Y",no="N")
a$prevalentStroke=as.factor(a$prevalentStroke)

a$prevalentHyp=ifelse(test=a$prevalentHyp==1,yes="Y",no="N")
a$prevalentHyp=as.factor(a$prevalentHyp)

a$diabetes=ifelse(test=a$diabetes==1,yes="Y",no="N")
a$diabetes=as.factor(a$diabetes)

a$TenYearCHD=ifelse(test=a$TenYearCHD==1,yes="Y",no="N")
a$TenYearCHD=as.factor(a$TenYearCHD)

str(a)
```

```
## tibble [4,238 x 16] (S3: tbl_df/tbl/data.frame)
##  $ Gender         : Factor w/ 2 levels "F","M": 2 1 2 1 1 1 1 1 2 2 ...
##  $ age            : num [1:4238] 39 46 48 61 46 43 63 45 52 43 ...
##  $ education (new): num [1:4238] 4 2 1 3 3 2 1 2 1 1 ...
##  $ currentSmoker  : Factor w/ 2 levels "N","Y": 1 1 2 2 2 1 1 2 1 2 ...
##  $ cigsPerDay(new): num [1:4238] 0 0 20 30 23 0 0 20 0 30 ...
##  $ BPMeds(new)    : Factor w/ 2 levels "N","Y": 1 1 1 1 1 1 1 1 1 1 ...
##  $ prevalentStroke: Factor w/ 2 levels "N","Y": 1 1 1 1 1 1 1 1 1 1 ...
##  $ prevalentHyp   : Factor w/ 2 levels "N","Y": 1 1 1 2 1 2 1 1 2 2 ...
##  $ diabetes       : Factor w/ 2 levels "N","Y": 1 1 1 1 1 1 1 1 1 1 ...
##  $ totChol(new)   : num [1:4238] 195 250 245 225 285 228 205 313 260 225 ...
##  $ sysBP          : num [1:4238] 106 121 128 150 130 ...
##  $ diaBP          : num [1:4238] 70 81 80 95 84 110 71 71 89 107 ...
##  $ BMI(new)       : num [1:4238] 27 28.7 25.3 28.6 23.1 ...
##  $ heartRate(new) : num [1:4238] 80 95 75 65 85 77 60 79 76 93 ...
##  $ glucose(new)   : num [1:4238] 77 76 70 103 85 99 85 78 79 88 ...
##  $ TenYearCHD     : Factor w/ 2 levels "N","Y": 1 1 1 2 1 1 2 1 1 1 ...
```

The unnecessary columns are dropped from the dataset and the categorical variables are properly mentioned using if else and as.factor commands. This will help the machine to understand that these are categorical variables and not any other numeric data.

```
#naming the variables

y=a$TenYearCHD #response variable converted to factors

#covariates
x1=a$Gender
x2=a$age
x3=as.factor(a$`education (new)`)
x4=a$currentSmoker
x5=a$`cigsPerDay(new)`
x6=a$`BPMeds(new)`
x7=a$prevalentStroke
x8=a$prevalentHyp
x9=a$diabetes
x10=a$`totChol(new)`
x11=a$sysBP
x12=a$diaBP
x13=a$`BMI(new)`
x14=a$`heartRate(new)`
x15=a$`glucose(new)`

df=data.frame(y,x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15)
View(df)
str(df)
```

```
## 'data.frame':    4238 obs. of  16 variables:
##  $ y  : Factor w/ 2 levels "N","Y": 1 1 1 2 1 1 2 1 1 1 ...
##  $ x1 : Factor w/ 2 levels "F","M": 2 1 2 1 1 1 1 1 2 2 ...
##  $ x2 : num  39 46 48 61 46 43 63 45 52 43 ...
##  $ x3 : Factor w/ 4 levels "1","2","3","4": 4 2 1 3 3 2 1 2 1 1 ...
##  $ x4 : Factor w/ 2 levels "N","Y": 1 1 2 2 2 1 1 2 1 2 ...
##  $ x5 : num  0 0 20 30 23 0 0 20 0 30 ...
##  $ x6 : Factor w/ 2 levels "N","Y": 1 1 1 1 1 1 1 1 1 1 ...
##  $ x7 : Factor w/ 2 levels "N","Y": 1 1 1 1 1 1 1 1 1 1 ...
##  $ x8 : Factor w/ 2 levels "N","Y": 1 1 1 2 1 2 1 1 2 2 ...
##  $ x9 : Factor w/ 2 levels "N","Y": 1 1 1 1 1 1 1 1 1 1 ...
##  $ x10: num  195 250 245 225 285 228 205 313 260 225 ...
##  $ x11: num  106 121 128 150 130 ...
##  $ x12: num  70 81 80 95 84 110 71 71 89 107 ...
##  $ x13: num  27 28.7 25.3 28.6 23.1 ...
##  $ x14: num  80 95 75 65 85 77 60 79 76 93 ...
##  $ x15: num  77 76 70 103 85 99 85 78 79 88 ...
```

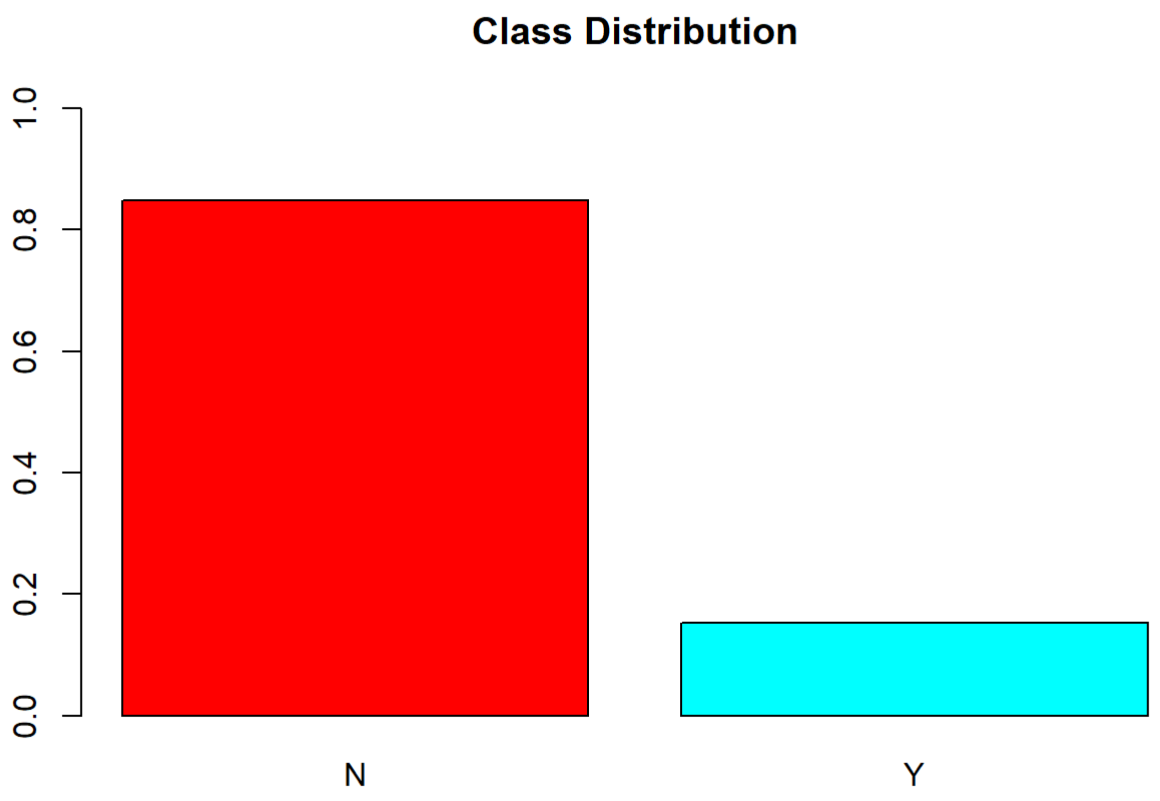
```
dim(df)
```

```
## [1] 4238   16
```

```
summary(df)
```

```
##  y        x1           x2          x3        x4            x5            x6      
##  N:3594   F:2419   Min.   :32.00   1:1720   N:2144   Min.   : 0.000   N:4114  
##  Y: 644   M:1819   1st Qu.:42.00   2:1358   Y:2094   1st Qu.: 0.000   Y: 124  
##                    Median :49.00   3: 687            Median : 0.000           
##                    Mean   :49.58   4: 473            Mean   : 9.003           
##                    3rd Qu.:56.00                     3rd Qu.:20.000           
##                    Max.   :70.00                     Max.   :70.000           
##  x7       x8       x9            x10             x11             x12        
##  N:4213   N:2922   N:4129   Min.   :107.0   Min.   : 83.5   Min.   : 48.00  
##  Y:  25   Y:1316   Y: 109   1st Qu.:206.0   1st Qu.:117.0   1st Qu.: 75.00  
##                             Median :234.0   Median :128.0   Median : 82.00  
##                             Mean   :236.7   Mean   :132.4   Mean   : 82.89  
##                             3rd Qu.:262.0   3rd Qu.:144.0   3rd Qu.: 89.88  
##                             Max.   :696.0   Max.   :295.0   Max.   :142.50  
##       x13             x14              x15        
##  Min.   :15.54   Min.   : 44.00   Min.   : 40.00  
##  1st Qu.:23.08   1st Qu.: 68.00   1st Qu.: 72.00  
##  Median :25.41   Median : 75.00   Median : 80.00  
##  Mean   :25.80   Mean   : 75.88   Mean   : 81.97  
##  3rd Qu.:28.04   3rd Qu.: 83.00   3rd Qu.: 85.00  
##  Max.   :56.80   Max.   :143.00   Max.   :394.00  
```

```
barplot(prop.table(table(df$y)),
        col=rainbow(2),
        ylim=c(0,1),
        main="Class Distribution") #representing the class imbalance
```



We have renamed the variables and checked for data imbalance. As we can observe there is high data imbalance in our dataset. The values belonging to the N class is far higher as compared to that of the Y class.

```
#dividing the dataset into training and testing datsets
set.seed(123)
ind<-sample(2,nrow(df),replace = TRUE,prob=c(0.7,0.3))
train<-df[ind==1,]
test<-df[ind==2,]

table(train$y)
```

```
##
##    N    Y
## 2516  451
```

```
prop.table(table(train$y)) #showing data imbalance in train data
```

```
##
##         N         Y
## 0.8479946 0.1520054
```

The entire dataset is divided into training and testing dataset in the ratio of 70:30. Even after splitting of dataset, we can see that there is high margin of difference in proportion of classes.

We have used over sampling, under sampling, both over and under sampling and synthetic data techniques to solve the issue of data imbalance. The data size is varied for different techniques.

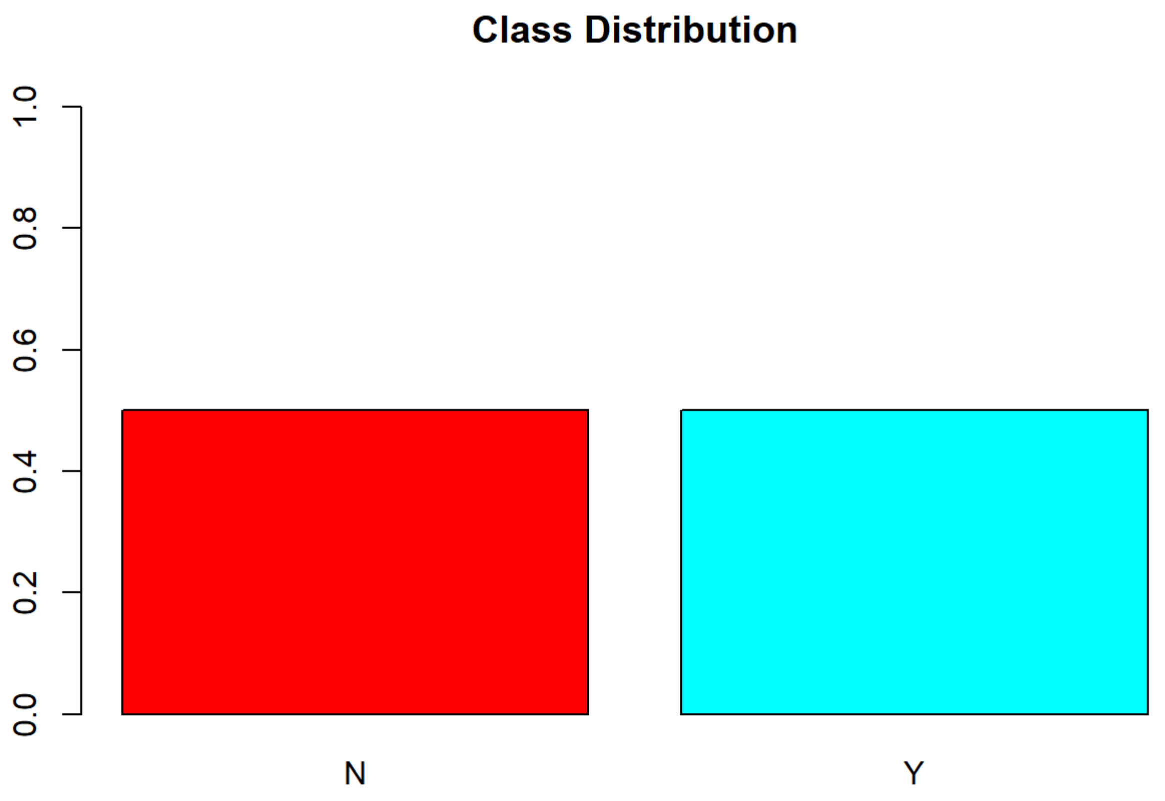
```
#oversampling
over<-ovun.sample(y~.,data = train, method = "over",N=5032)$data
dim(over)
```

```
## [1] 5032   16
```

```
prop.table(table(over$y))
```

```
##
##   N   Y
## 0.5 0.5
```

```
barplot(prop.table(table(over$y)),
        col=rainbow(2),
        ylim=c(0,1),
        main="Class Distribution")
```



As we can see that, that the huge margin of imbalance is solved. In this technique of over sampling, samples are chosen randomly with repitition from the non dominant class (Y in this case) until the data imbalance issue is resolved.

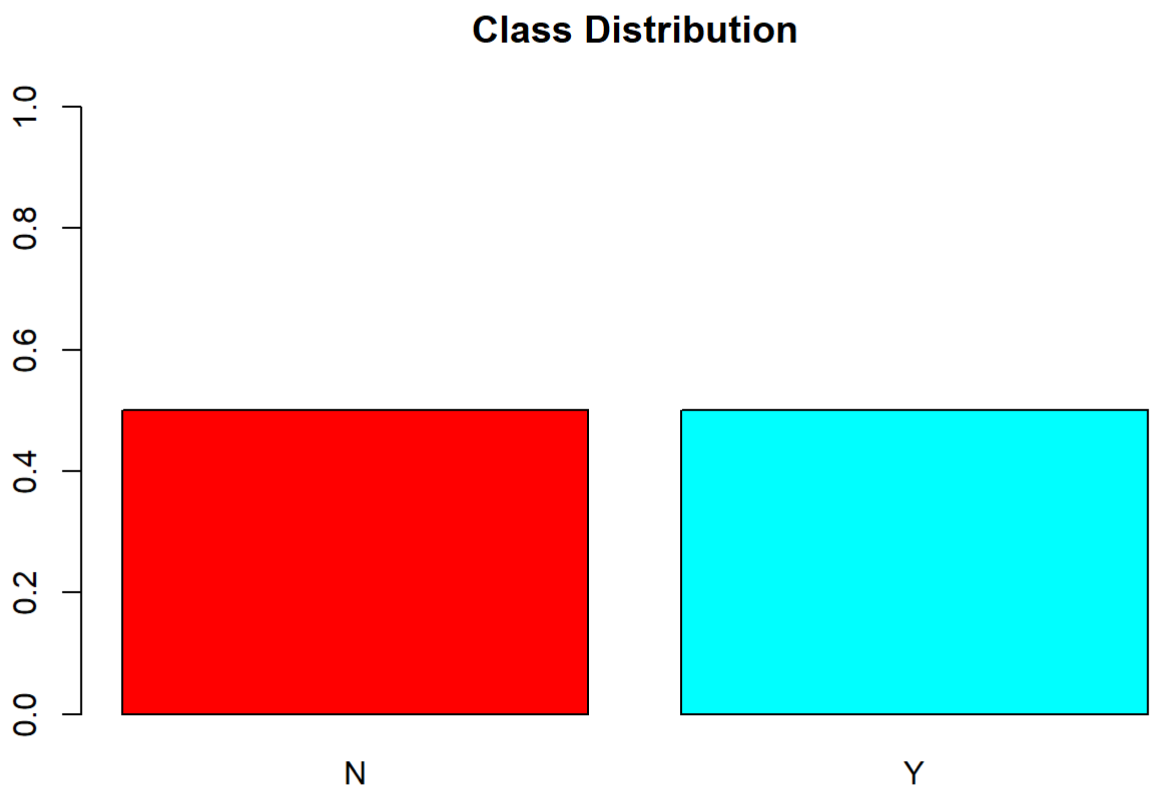
```
#undersampling
under<-ovun.sample(y~.,data = train,method = "under",N=902)$data
dim(under)
```

```
## [1] 902  16
```

```
prop.table(table(under$y))
```

```
## 
##   N   Y 
## 0.5 0.5
```

```
barplot(prop.table(table(under$y)),
        col=rainbow(2),
        ylim=c(0,1),
        main="Class Distribution")
```



As we can see that, that the huge margin of imbalance is solved. In this technique of under sampling, samples are chosen randomly with repitition from the dominant class (N in this case) until the data imbalance issue is resolved.

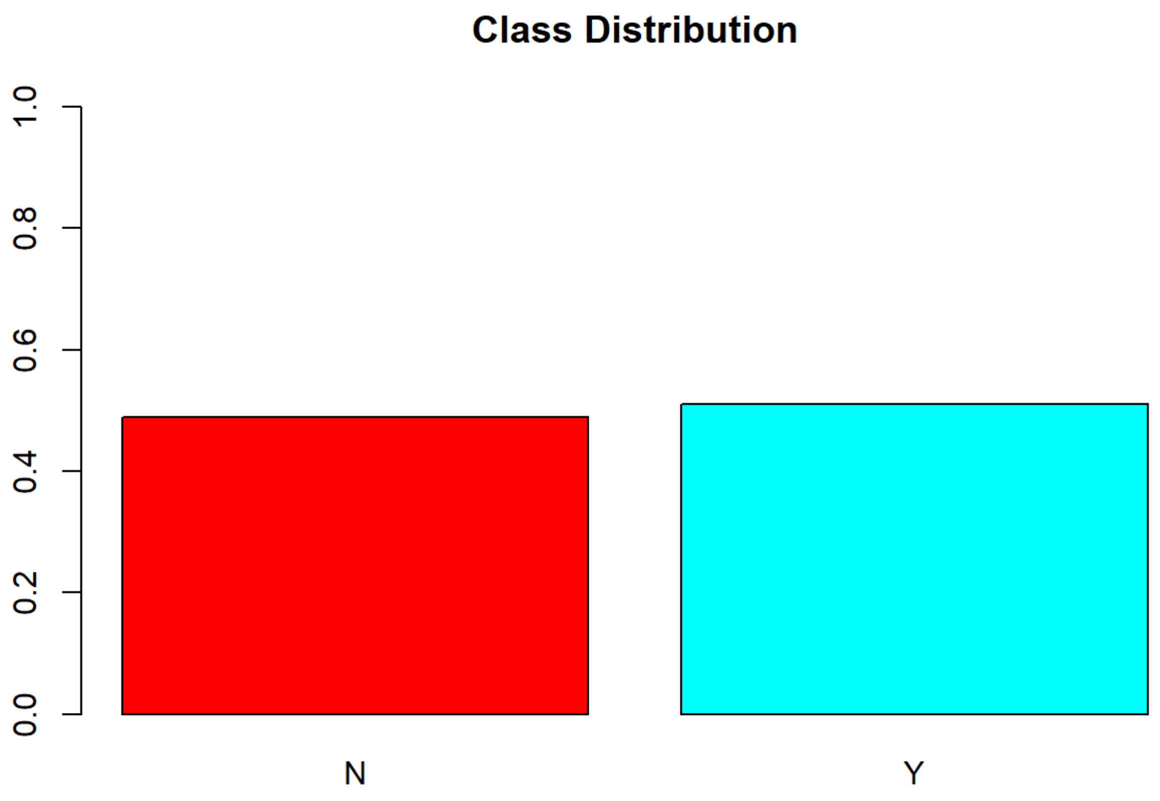
```
#both over and under sampling
both<-ovun.sample(y~.,data = train,method = "both",p=0.5,seed = 222,N=2967)$data
dim(both)
```

```
## [1] 2967   16
```

```
prop.table(table(both$y))
```

```
## 
##         N         Y 
## 0.4890462 0.5109538
```

```
barplot(prop.table(table(both$y)),
        col=rainbow(2),
        ylim=c(0,1),
        main="Class Distribution")
```



As we can see that, that the huge margin of imbalance is solved. In this technique of both over and under sampling, samples are chosen randomly with repitition from both the dominant and non dominant classes until the data imbalance issue is resolved.

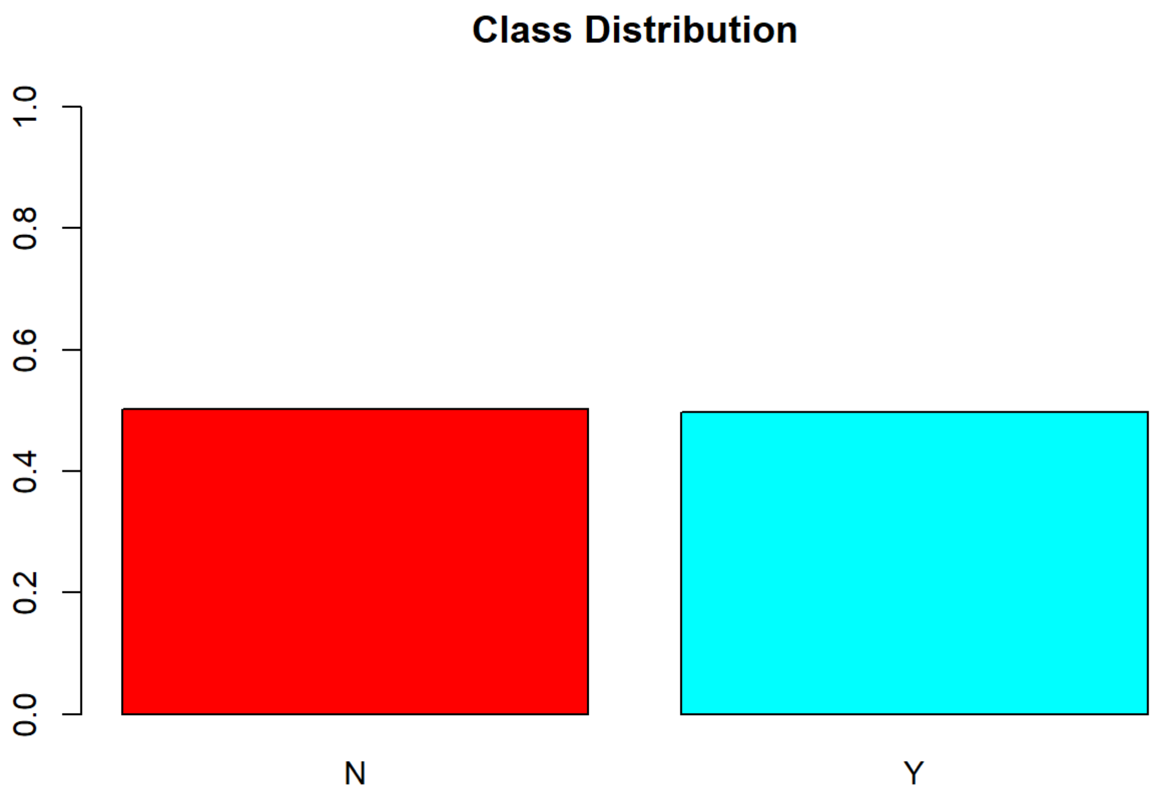
```
#synthetic data generation
syndata<-ROSE(y~.,data = train,N=4000,seed = 111)$data
dim(syndata)
```

```
## [1] 4000   16
```

```
prop.table(table(syndata$y))
```

```
##
##      N      Y
## 0.5025 0.4975
```

```
barplot(prop.table(table(syndata$y)),
        col=rainbow(2),
        ylim=c(0,1),
        main="Class Distribution")
```



As we can see that, that the huge margin of imbalance is solved. In this technique of synthetic data generation, the machine studies the data completely and generates data on it's own until the data imbalance issue is resolved.

Now that, the problem of data imbalance is solved, we can finally move on to our model building and compare the results between them.

Let's begin with model building with our over sampling data using Logistic Regression

```
#model building with oversampling data
model1=glm(y~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10+x11+x12+x13+x14+x15,data = over,family = binomial
(link = "logit"))

#checking for multicollinearity
vif(model1) #no multicollinearity present
```

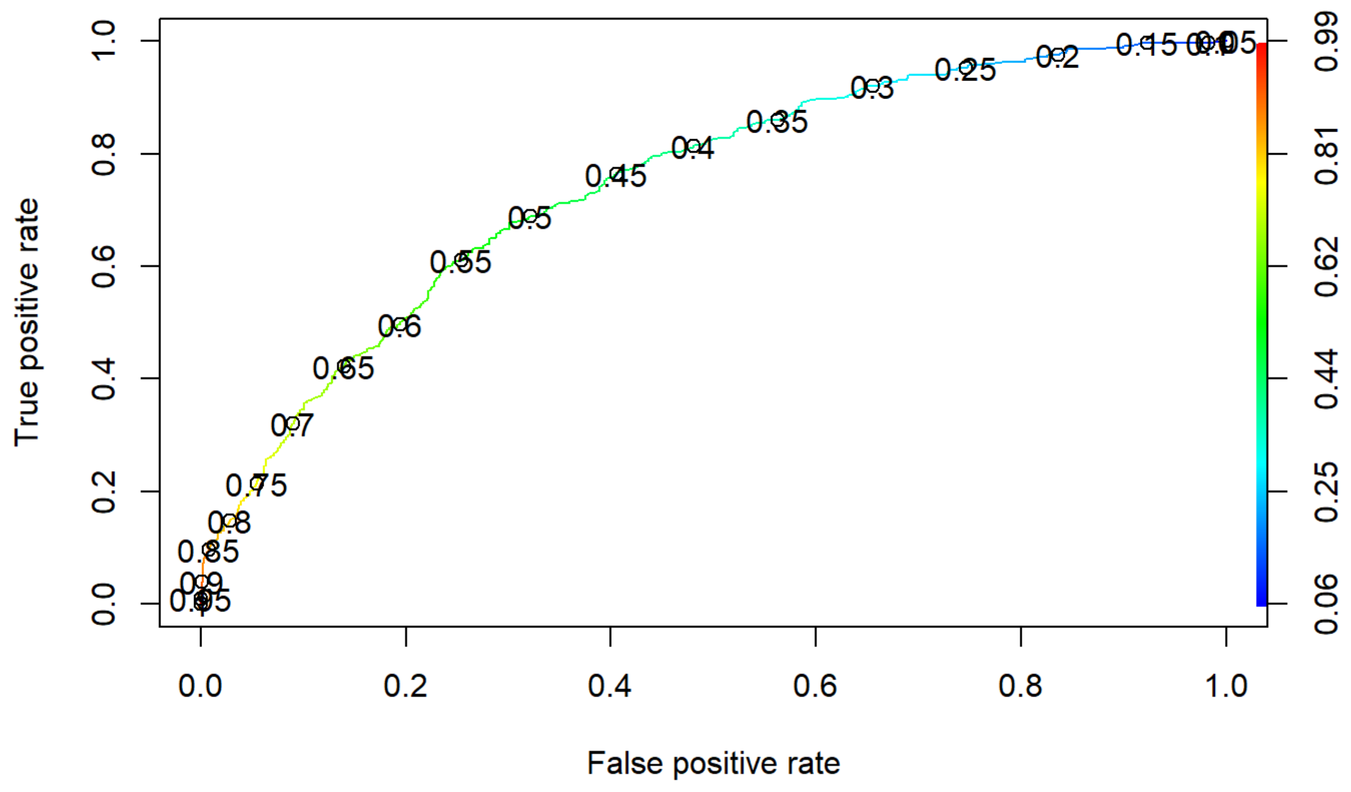
```
##         GVIF Df GVIF^(1/(2*Df))
## x1  1.263809  1        1.124193
## x2  1.357848  1        1.165267
## x3  1.130549  3        1.020661
## x4  2.703972  1        1.644376
## x5  2.836534  1        1.684201
## x6  1.089886  1        1.043976
## x7  1.015634  1        1.007787
## x8  2.087134  1        1.444692
## x9  1.610370  1        1.269004
## x10 1.064832  1        1.031907
## x11 3.405535  1        1.845409
## x12 2.779315  1        1.667128
## x13 1.174203  1        1.083607
## x14 1.094558  1        1.046211
## x15 1.637861  1        1.279789
```

```
#obtaining ideal cutoff value
trainpred<-predict(model1,over,type = "response")
length(trainpred)
```

```
## [1] 5032
```

```
ROCRPred<-prediction(trainpred,over$y)
ROCRPerf<-performance(ROCRPred,"tpr","fpr")

plot(ROCRPerf,colorize=TRUE,print.cutoffs.at=seq(0,1,0.05))
```



```
#0.5 seems to be an ideal cutoff value
```

We can choose 0.5 as our optimum cutoff value as corresponding to 0.5, our true positive rate is a bit over 0.6 and the false positive rate is a bit around 0.3 which is very less. Choosing this point as our cutoff may give good results.

Now, moving on to the final predictions and generating the confusion matrix for evaluation of our model.

```
#performing prediction using test data
testpredlr_over<-predict(model1,test,type = "response")
length(testpredlr_over)
```

```
## [1] 1271
```

```
testpredlr_over=ifelse(testpredlr_over>=0.5,yes = "Y",no="N")
testpredlr_over=as.factor(testpredlr_over)

#creating confusion matrix of the prediction
ConfusionMatrix(testpredlr_over,test$y)
```

```
##       y_pred
## y_true   N   Y
##      N 726 352
##      Y  70 123
```

```
acc_lr_over=Accuracy(testpredlr_over,test$y)*100
prec_lr_over=Precision(test$y,testpredlr_over,positive = "Y")*100
rec_lr_over=Recall(test$y,testpredlr_over,positive = "Y")*100
spec_lr_over=Specificity(test$y,testpredlr_over,positive = "Y")*100
f1_lr_over=F1_Score(test$y,testpredlr_over,positive = "Y")*100
```

Now, we have created a Logistic Regression model using the under sampling dataset.

```
#model building with undersampling data
model2=glm(y~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10+x11+x12+x13+x14+x15,data = under,family = binomia
l(link = "logit"))

#checking for multicollinearity
vif(model2) #no multicollinearity present
```

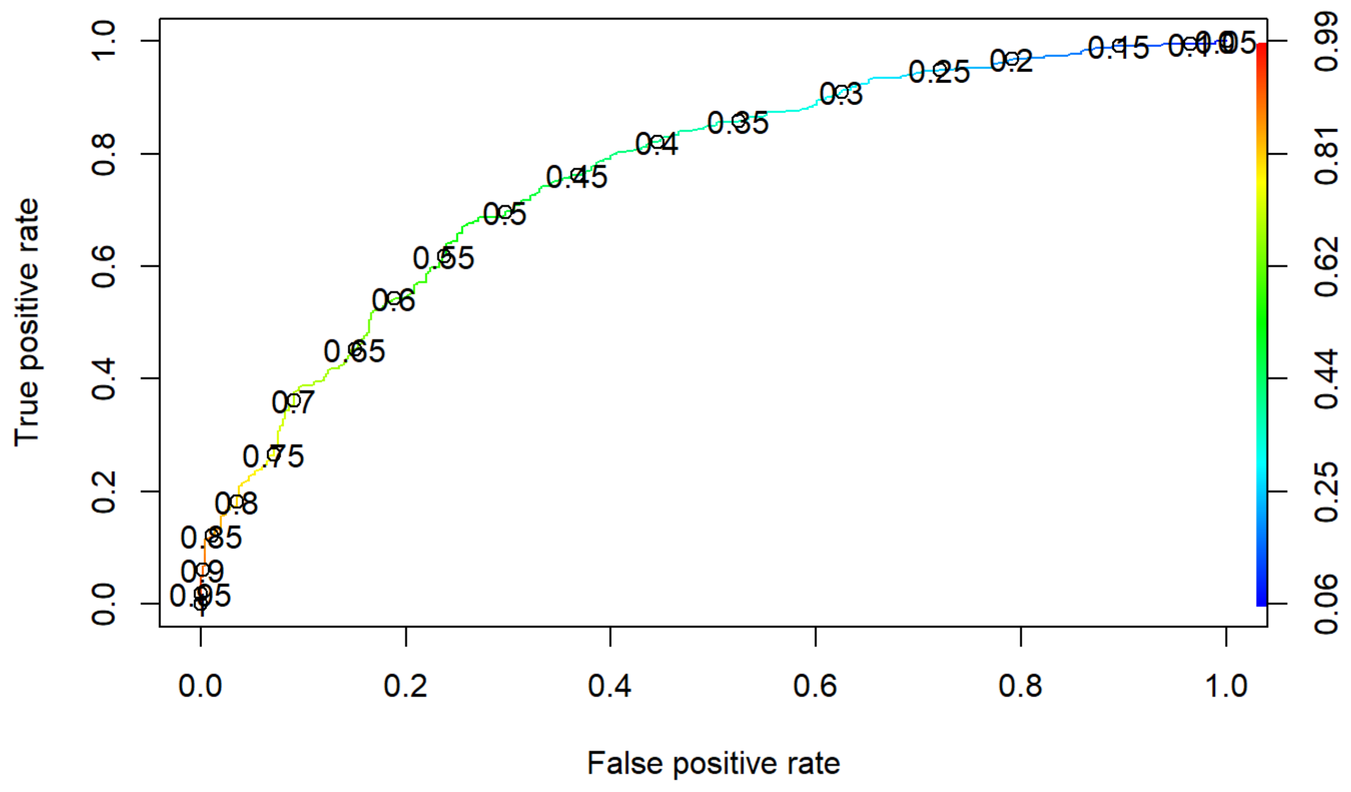
```
##         GVIF Df GVIF^(1/(2*Df))
## x1  1.268056  1        1.126080
## x2  1.374452  1        1.172370
## x3  1.161822  3        1.025313
## x4  2.763382  1        1.662342
## x5  2.825906  1        1.681043
## x6  1.077499  1        1.038026
## x7  1.025306  1        1.012574
## x8  2.059181  1        1.434985
## x9  1.538419  1        1.240330
## x10 1.051246  1        1.025303
## x11 3.387446  1        1.840502
## x12 2.711226  1        1.646580
## x13 1.179162  1        1.085892
## x14 1.086709  1        1.042453
## x15 1.557270  1        1.247906
```

```
#obtaining ideal cutoff value
trainpred<-predict(model2,under,type = "response")
length(trainpred)
```

```
## [1] 902
```

```
ROCRPred<-prediction(trainpred,under$y)
ROCRPerf<-performance(ROCRPred,"tpr","fpr")

plot(ROCRPerf,colorize=TRUE,print.cutoffs.at=seq(0,1,0.05))
```



```
#0.5 seems to be an ideal cutoff value
```

We can choose 0.5 as our optimum cutoff value as corresponding to 0.5, our true positive rate is a bit over 0.6 and the false positive rate is a bit around 0.3 which is very less. Choosing this point as our cutoff may give good results.

Now, moving on to the final predictions and generating the confusion matrix for evaluation of our model.

```
#performing prediction using test data
testpredlr_under<-predict(model2,test,type = "response")
length(testpredlr_under)
```

```
## [1] 1271
```

```
testpredlr_under=ifelse(testpredlr_under>=0.5,yes = "Y",no="N")
testpredlr_under=as.factor(testpredlr_under)

#creating confusion matrix of the prediction
ConfusionMatrix(testpredlr_under,test$y)
```

```
##       y_pred
## y_true   N   Y
##      N 717 361
##      Y  69 124
```

```
acc_lr_under=Accuracy(testpredlr_under,test$y)*100
prec_lr_under=Precision(test$y,testpredlr_under,positive = "Y")*100
rec_lr_under=Recall(test$y,testpredlr_under,positive = "Y")*100
spec_lr_under=Specificity(test$y,testpredlr_under,positive = "Y")*100
f1_lr_under=F1_Score(test$y,testpredlr_under,positive = "Y")*100
```

Now, we have created a Logistic Regression model using the both over and under sampling dataset.

```
#model building with both over and under sampling data
model3=glm(y~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10+x11+x12+x13+x14+x15,data = both,family =binomial
(link = "logit"))

#checking for multicollinearity
vif(model3) #no multicollinearity present
```

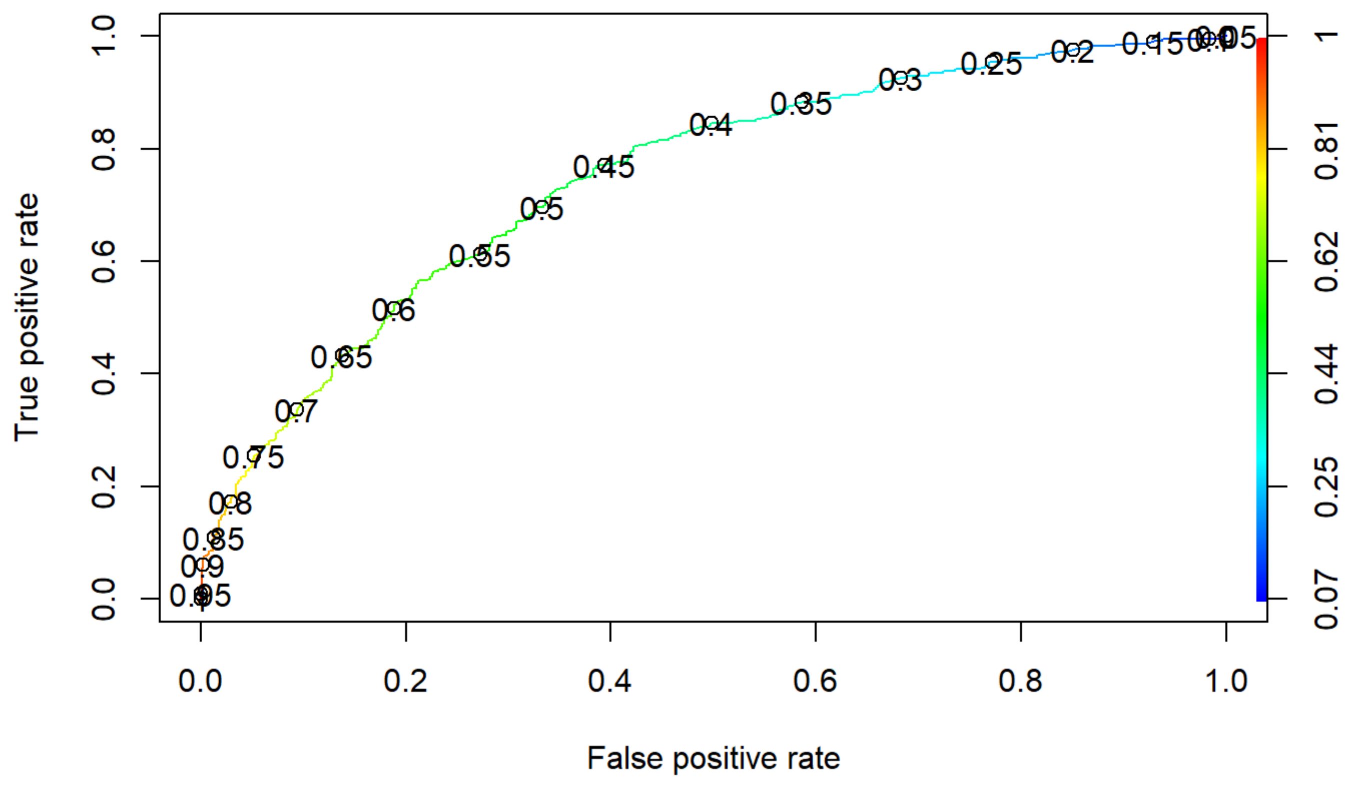
```
##         GVIF Df GVIF^(1/(2*Df))
## x1  1.279898  1        1.131326
## x2  1.341973  1        1.158436
## x3  1.159616  3        1.024989
## x4  2.515237  1        1.585950
## x5  2.654631  1        1.629304
## x6  1.070045  1        1.034430
## x7  1.016682  1        1.008307
## x8  2.029955  1        1.424765
## x9  1.803845  1        1.343073
## x10 1.059190  1        1.029170
## x11 3.155014  1        1.776236
## x12 2.546101  1        1.595651
## x13 1.148832  1        1.071836
## x14 1.100659  1        1.049123
## x15 1.808439  1        1.344782
```

```
#obtaining ideal cutoff value
trainpred<-predict(model3,both,type = "response")
length(trainpred)
```

```
## [1] 2967
```

```
ROCRPred<-prediction(trainpred,both$y)
ROCRPerf<-performance(ROCRPred,"tpr","fpr")

plot(ROCRPerf,colorize=TRUE,print.cutoffs.at=seq(0,1,0.05))
```



```
#0.5 seems to be an ideal cutoff value
```

We can choose 0.5 as our optimum cutoff value as corresponding to 0.5, our true positive rate is a bit over 0.6 and the false positive rate is around 0.3 which is very less. Choosing this point as our cutoff may give good results.

Now, moving on to the final predictions and generating the confusion matrix for evaluation of our model.

```
#performing prediction using test data
testpredlr_both<-predict(model3,test,type = "response")
length(testpredlr_both)
```

```
## [1] 1271
```

```
testpredlr_both=ifelse(testpredlr_both>=0.5,yes = "Y",no="N")
testpredlr_both=as.factor(testpredlr_both)

#creating confusion matrix of the prediction
ConfusionMatrix(testpredlr_both,test$y)
```

```
##       y_pred
## y_true   N   Y
##      N 731 347
##      Y  71 122
```

```
acc_lr_both=Accuracy(testpredlr_both,test$y)*100
prec_lr_both=Precision(test$y,testpredlr_both,positive = "Y")*100
rec_lr_both=Recall(test$y,testpredlr_both,positive = "Y")*100
spec_lr_both=Specificity(test$y,testpredlr_both,positive = "Y")*100
f1_lr_both=F1_Score(test$y,testpredlr_both,positive = "Y")*100
```

Now, we have created a Logistic Regression model using the synthetic data dataset.

```
#model building with synthetically generated data
model4=glm(y~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10+x11+x12+x13+x14+x15,data = syndata,family = binom
ial(link = "logit"))

#checking for multicollinearity
vif(model4) #no multicollinearity present
```

```
##         GVIF Df GVIF^(1/(2*Df))
## x1  1.195178  1        1.093242
## x2  1.193210  1        1.092342
## x3  1.115678  3        1.018411
## x4  2.020571  1        1.421468
## x5  2.035715  1        1.426785
## x6  1.076986  1        1.037779
## x7  1.018342  1        1.009129
## x8  1.875451  1        1.369471
## x9  1.409728  1        1.187320
## x10 1.039768  1        1.019690
## x11 1.995823  1        1.412736
## x12 1.777913  1        1.333384
## x13 1.119012  1        1.057834
## x14 1.053279  1        1.026294
## x15 1.408756  1        1.186910
```

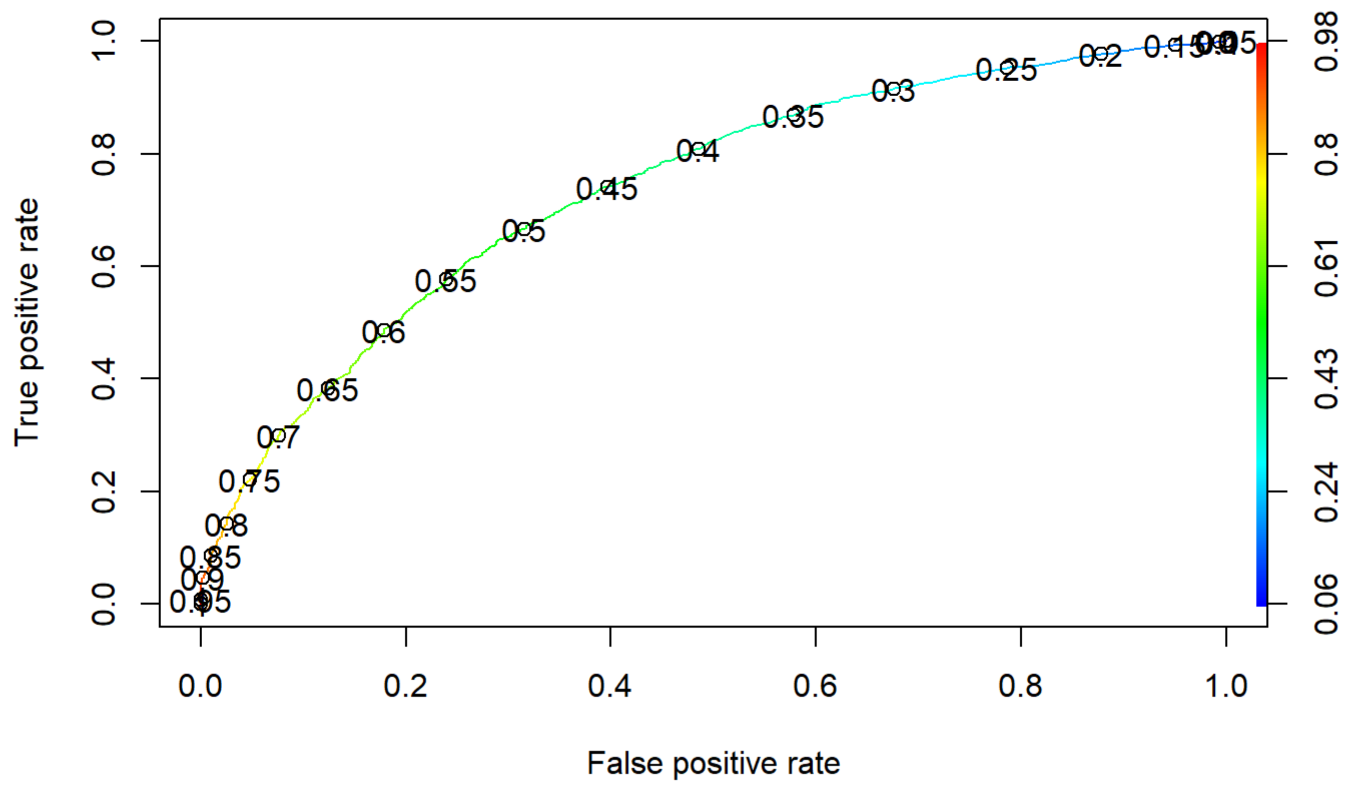
The model deviance is definitely on the higher end but there is no multicollinearity present.

```
#obtaining ideal cutoff value
trainpred<-predict(model4,syndata,type = "response")
length(trainpred)
```

```
## [1] 4000
```

```
ROCRPred<-prediction(trainpred,syndata$y)
ROCRPerf<-performance(ROCRPred,"tpr","fpr")

plot(ROCRPerf,colorize=TRUE,print.cutoffs.at=seq(0,1,0.05))
```



```
#0.5 seems to be an ideal cutoff value
```

We can choose 0.5 as our optimum cutoff value as corresponding to 0.5, our true positive rate is a bit over 0.6 and the false positive rate is around 0.3 which is very less. Choosing this point as our cutoff may give good results.

Now, moving on to the final predictions and generating the confusion matrix for evaluation of our model.

```
#performing prediction using test data
testpredlr_syn<-predict(model4,test,type = "response")
length(testpredlr_syn)
```

```
## [1] 1271
```

```
testpredlr_syn=ifelse(testpredlr_syn>=0.5,yes = "Y",no="N")
testpredlr_syn=as.factor(testpredlr_syn)

#creating confusion matrix of the prediction
ConfusionMatrix(testpredlr_syn,test$y)
```

```
##        y_pred
## y_true   N   Y
##       N 739 339
##       Y  76 117
```

```
acc_lr_syn=Accuracy(testpredlr_syn,test$y)*100
prec_lr_syn=Precision(test$y,testpredlr_syn,positive = "Y")*100
rec_lr_syn=Recall(test$y,testpredlr_syn,positive = "Y")*100
spec_lr_syn=Specificity(test$y,testpredlr_syn,positive = "Y")*100
f1_lr_syn=F1_Score(test$y,testpredlr_syn,positive = "Y")*100
```

We have created our predictive model using Logistic Regression using the four datasets that solves the issue of data imbalance.

Now, we move on to a non parametric approach called Random Forest. We begin with creating a predictive model using Random Forest with the over sampling dataset.

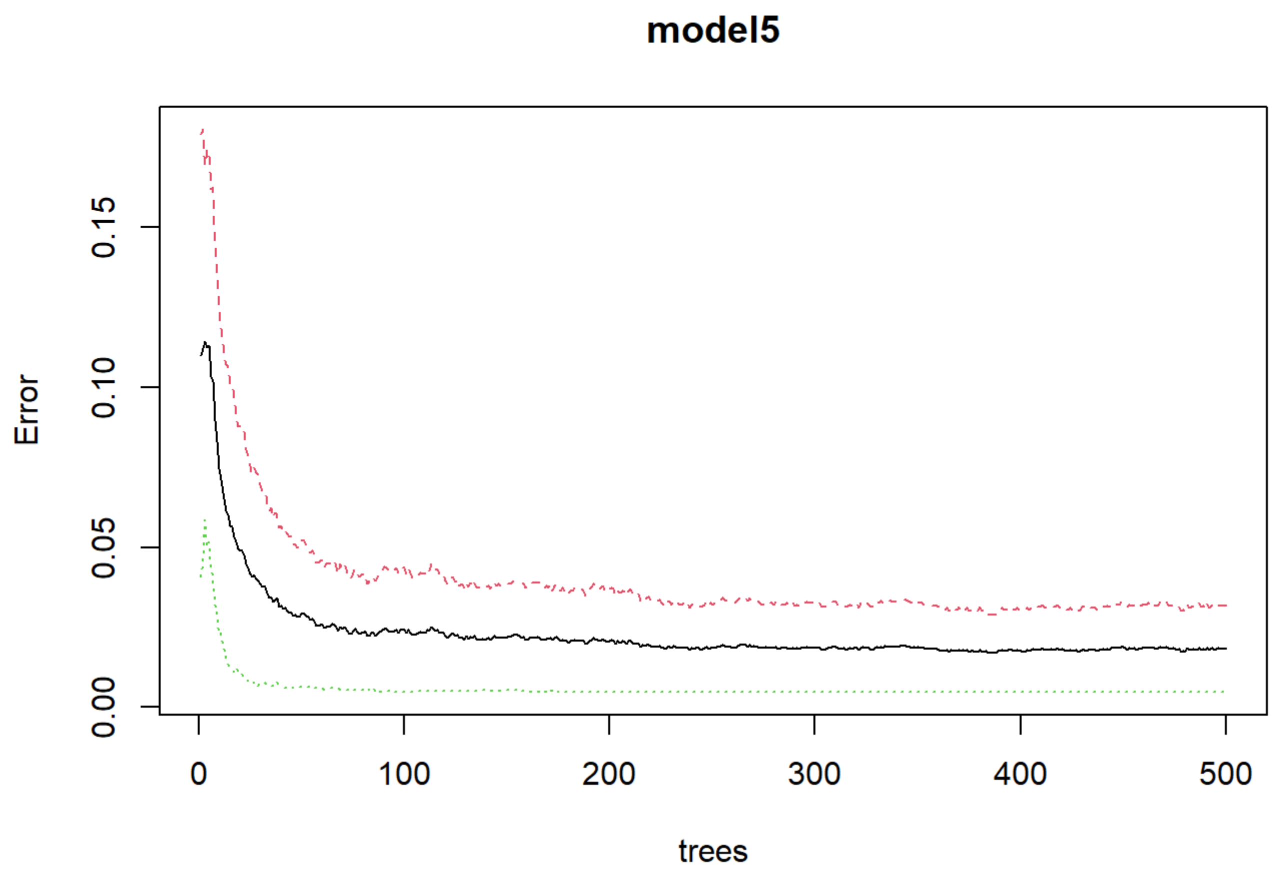
```
set.seed(100)
#building a primary RF model
model5<-randomForest(y~.,data=over)
model5
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = over)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 1.83%
## Confusion matrix:
##      N    Y class.error
## N 2436   80 0.031796502
## Y   12 2504 0.004769475
```

The OOB error estimate and the class errors are very less which indicates that our model fitting is very efficient. We can reach this conclusion from the confusion matrix as well.

Now, we can check for our optimum value for ntree and mtry from the following graphs and tune the model accordingly if necessary.

```
plot(model5)
```

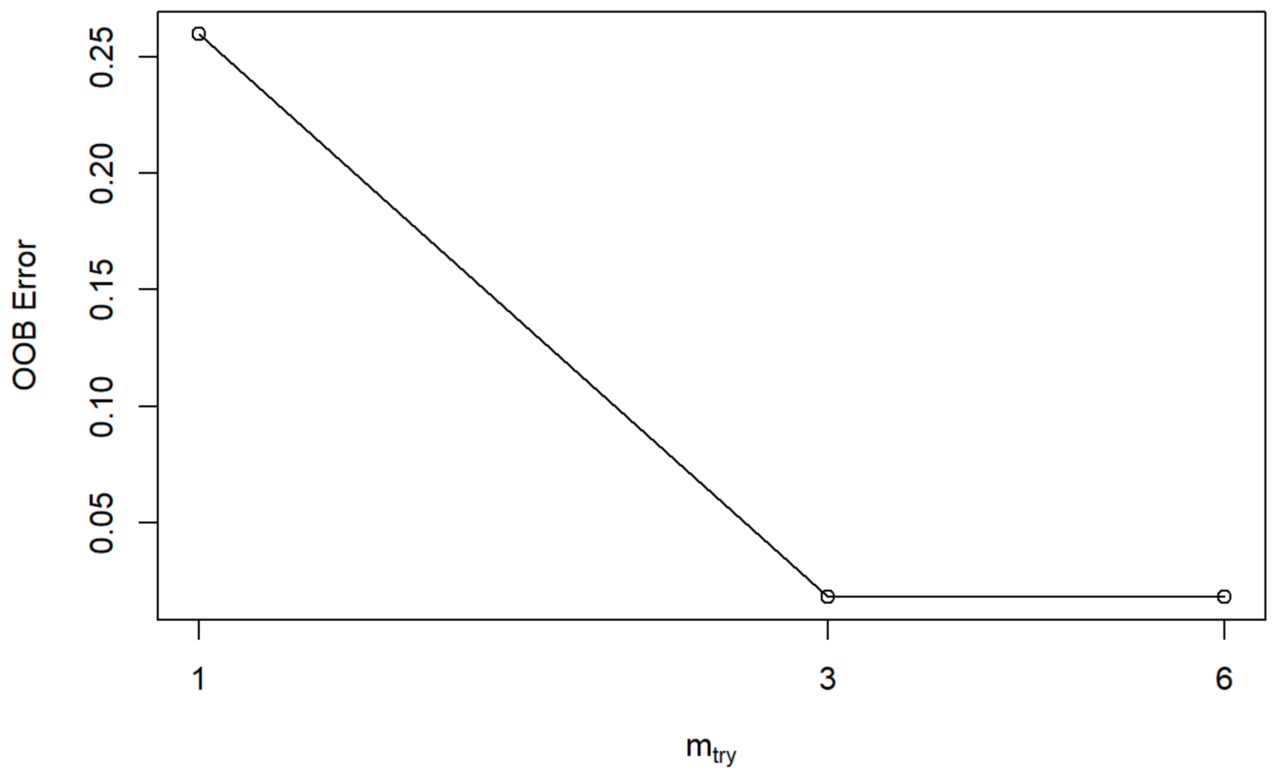


From the above graph, we can see that the error is almost saturated for 500 trees and hence, 500 seems to be an appropriate choice for ntree.

Now, let's check our optimum value for mtry.

```
tuneRF(over[,-1],over[,1],stepFactor = 0.5,plot = TRUE,ntreeTry = 500,trace = TRUE,improve =
0.05)
```

```
## mtry = 3  OOB error = 1.83%
## Searching left ...
## mtry = 6     OOB error = 1.83%
## 0 0.05
## Searching right ...
## mtry = 1     OOB error = 25.95%
## -13.19565 0.05
```



```
##       mtry   OOBError
## 1.OOB    1 0.25953895
## 3.OOB    3 0.01828299
## 6.OOB    6 0.01828299
```

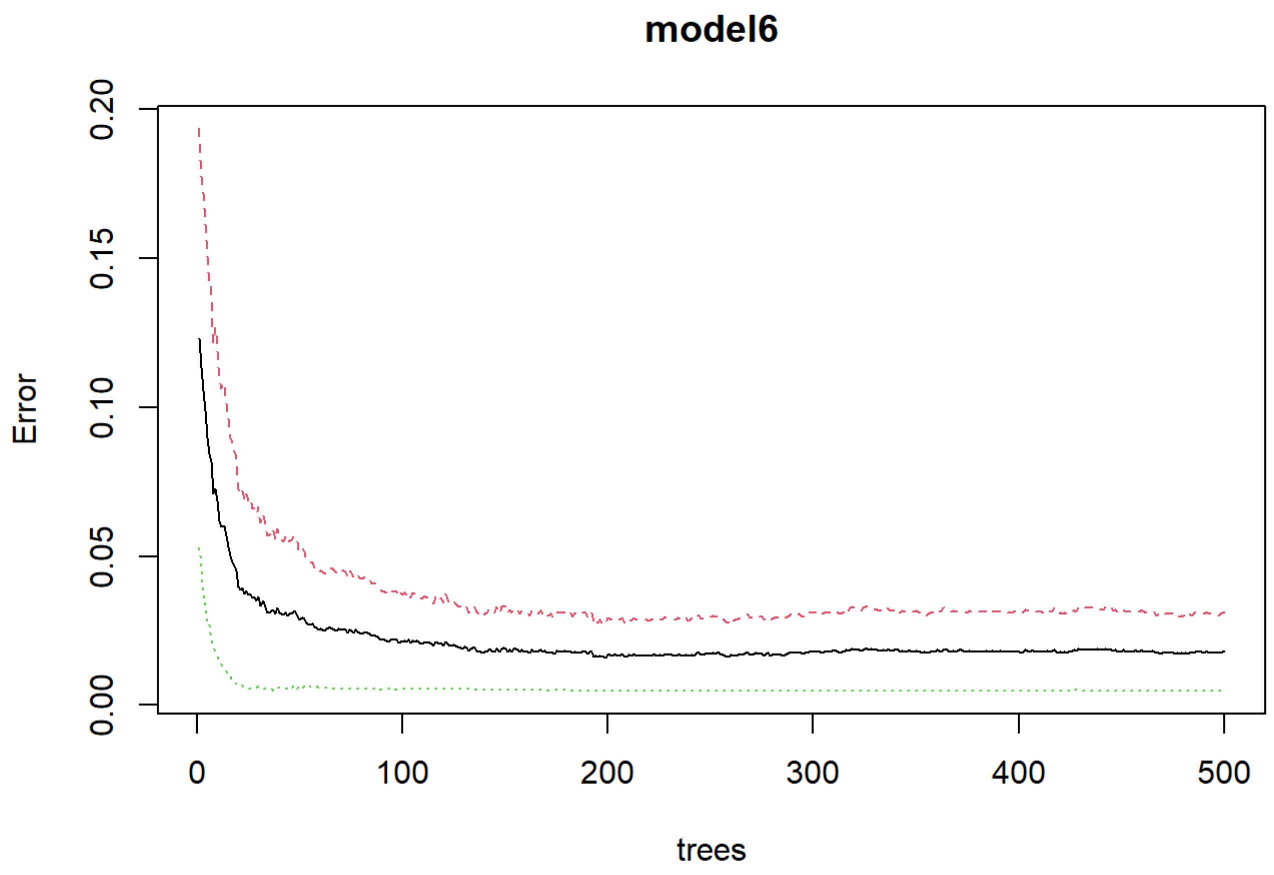
We can observe that the appropriate choice of mtry turns out to be 3, because for this value the OOB error estimate is the least. But on further experimentation of the appropriate value of mtry from 3 to 6, we found out that OOB error estimate is least for mtry=4.

Hence, we tune the parameters of our random forest model accordingly and then move on to performing the predictions.

```
set.seed(100)
model6<-randomForest(y~.,data = over,ntree=500,mtry=4, importance=TRUE,proximity=TRUE)
model6
```

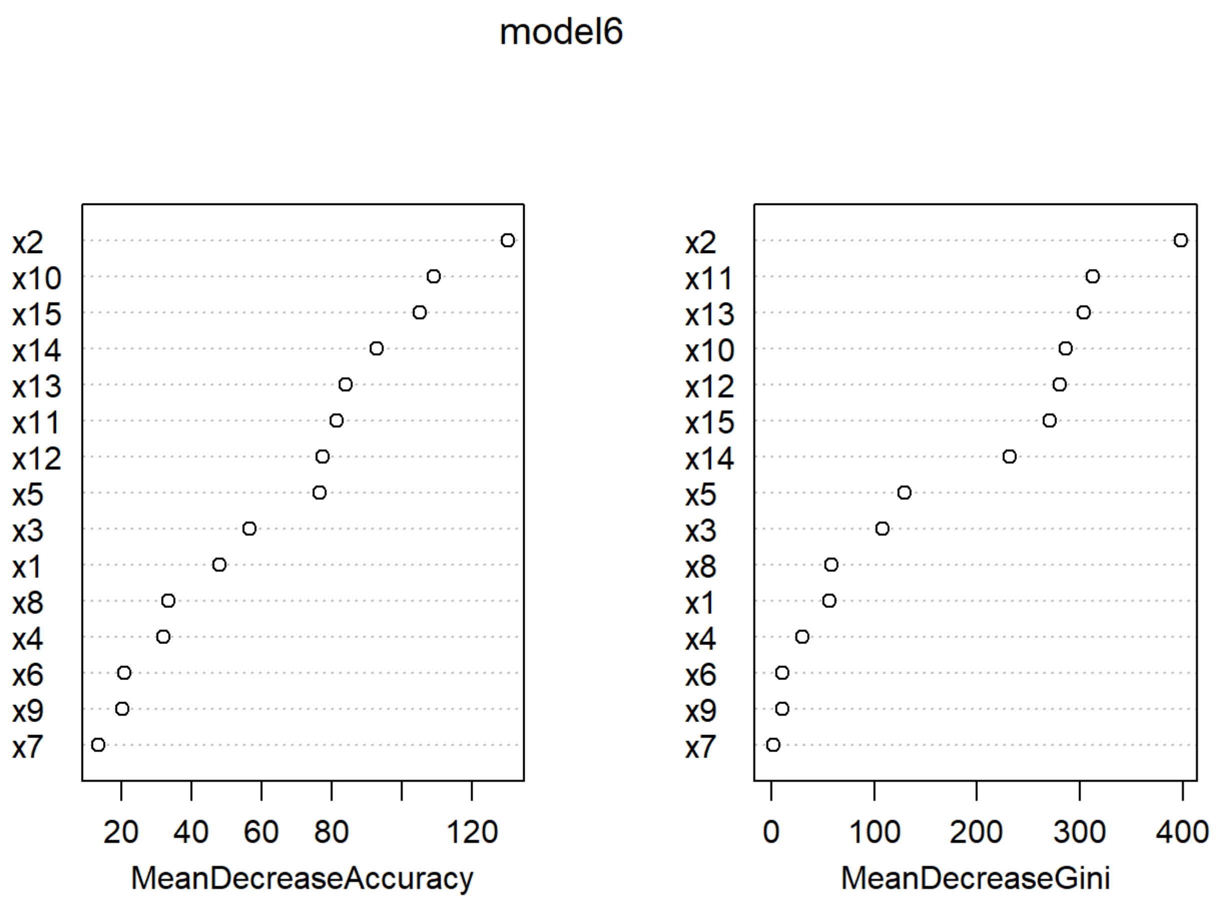
```
##
## Call:
##  randomForest(formula = y ~ ., data = over, ntree = 500, mtry = 4,      importance = TRUE, 
proximity = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 1.79%
## Confusion matrix:
##      N    Y class.error
## N 2438   78 0.031001590
## Y   12 2504 0.004769475
```

```
plot(model6)
```



This looks like a very efficient model. We can have a look about the important features in our predictive model.

```
#Variable importance
varImpPlot(model6,sort = TRUE) #this graph measures how pure the nodes are at the end of the
 tree without each variable
```



```
importance(model6)
```

```
##              N         Y MeanDecreaseAccuracy MeanDecreaseGini
## x1   5.1875890  48.32093             48.05609         57.06919
## x2  32.2383770 128.97590            130.27271        398.22916
## x3  -1.2302655  57.37409             56.64229        108.67387
## x4  -3.9507259  31.56239             31.89674         30.54912
## x5  -0.6454122  77.94450             76.63891        130.07972
## x6   3.5244848  21.52033             20.73454         11.24325
## x7   0.5845600  13.63033             13.29149          2.51714
## x8   3.1339095  33.82751             33.32894         58.17045
## x9   4.4323291  19.78235             20.11141         10.68581
## x10  8.1545890 112.36378            109.16078        286.62861
## x11 11.9515653  82.08805             81.55864        312.86899
## x12 18.8253743  76.65214             77.56076        280.48497
## x13  7.2924179  83.70835             83.97250        303.83395
## x14  6.3045824  94.28278             92.73447        231.59925
## x15  8.4800198 109.19971            105.30550        271.07067
```

```
varUsed(model6)
```

```
##  [1]  4956 27274 13049  3912 15722  1600   446  2982  1364 31806 30939 29474
## [13] 33136 27919 30114
```

Now, let's move on to performing predictions and creating the confusion matrix.

```
#prediction using test data
testpredrf_over<-predict(model6,test)
length(testpredrf_over)
```

```
## [1] 1271
```

```
#confusion matrix
ConfusionMatrix(testpredrf_over,test$y)
```

```
##       y_pred
## y_true    N    Y
##      N 1049   29
##      Y  171   22
```

```
acc_rf_over=Accuracy(testpredrf_over,test$y)*100
prec_rf_over=Precision(test$y,testpredrf_over,positive = "Y")*100
rec_rf_over=Recall(test$y,testpredrf_over,positive = "Y")*100
spec_rf_over=Specificity(test$y,testpredrf_over,positive = "Y")*100
f1_rf_over=F1_Score(test$y,testpredrf_over,positive = "Y")*100
```

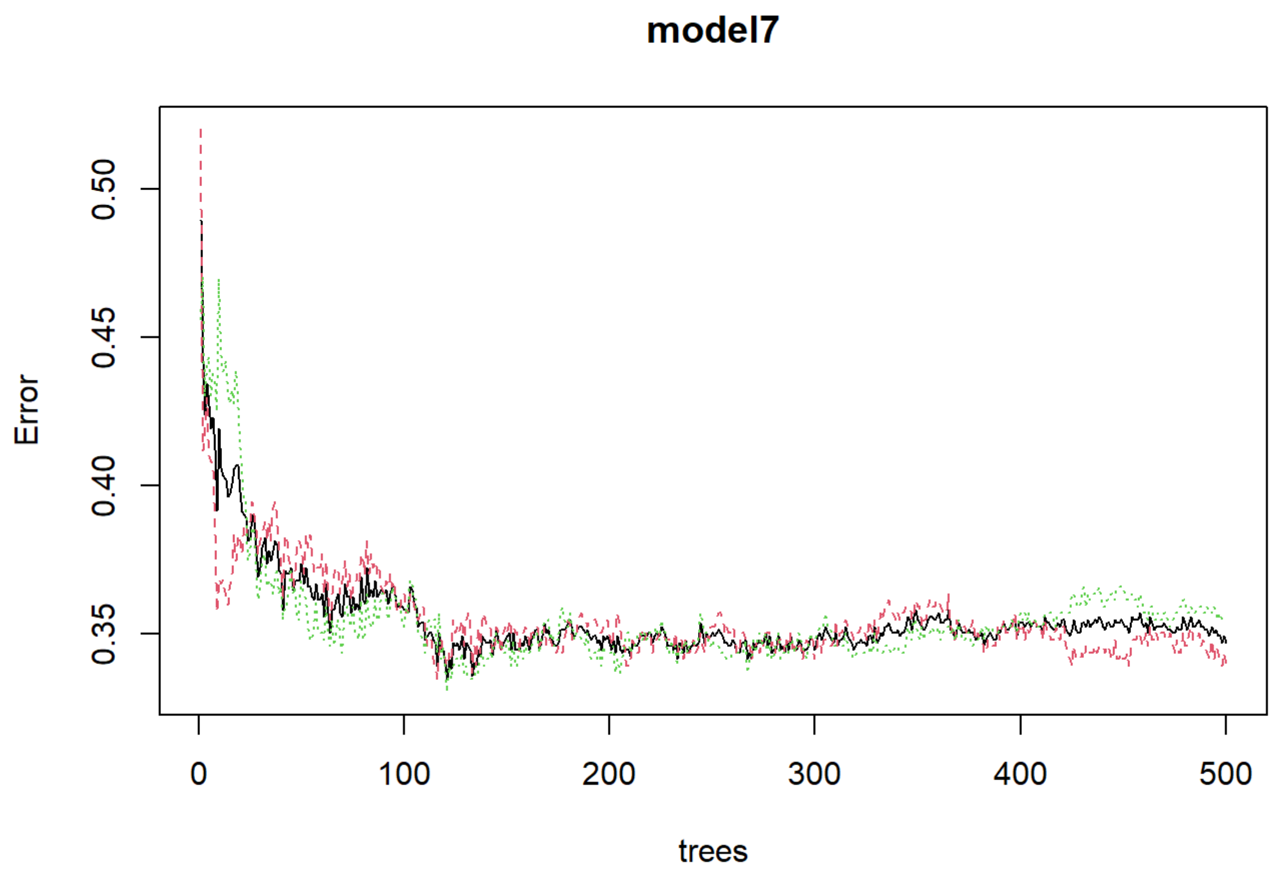
This is a clear case of overfitting as the Random Forest model based on over sampling data isn't providing proper predictions and the precision is quite low.

Now, we create a Random Forest model based on the under sampling dataset.

```
set.seed(100)
#building a primary RF model
model7<-randomForest(y~.,data=under)
model7
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = under)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 34.7%
## Confusion matrix:
##     N   Y class.error
## N 298 153   0.3392461
## Y 160 291   0.3547672
```

```
plot(model7)
```

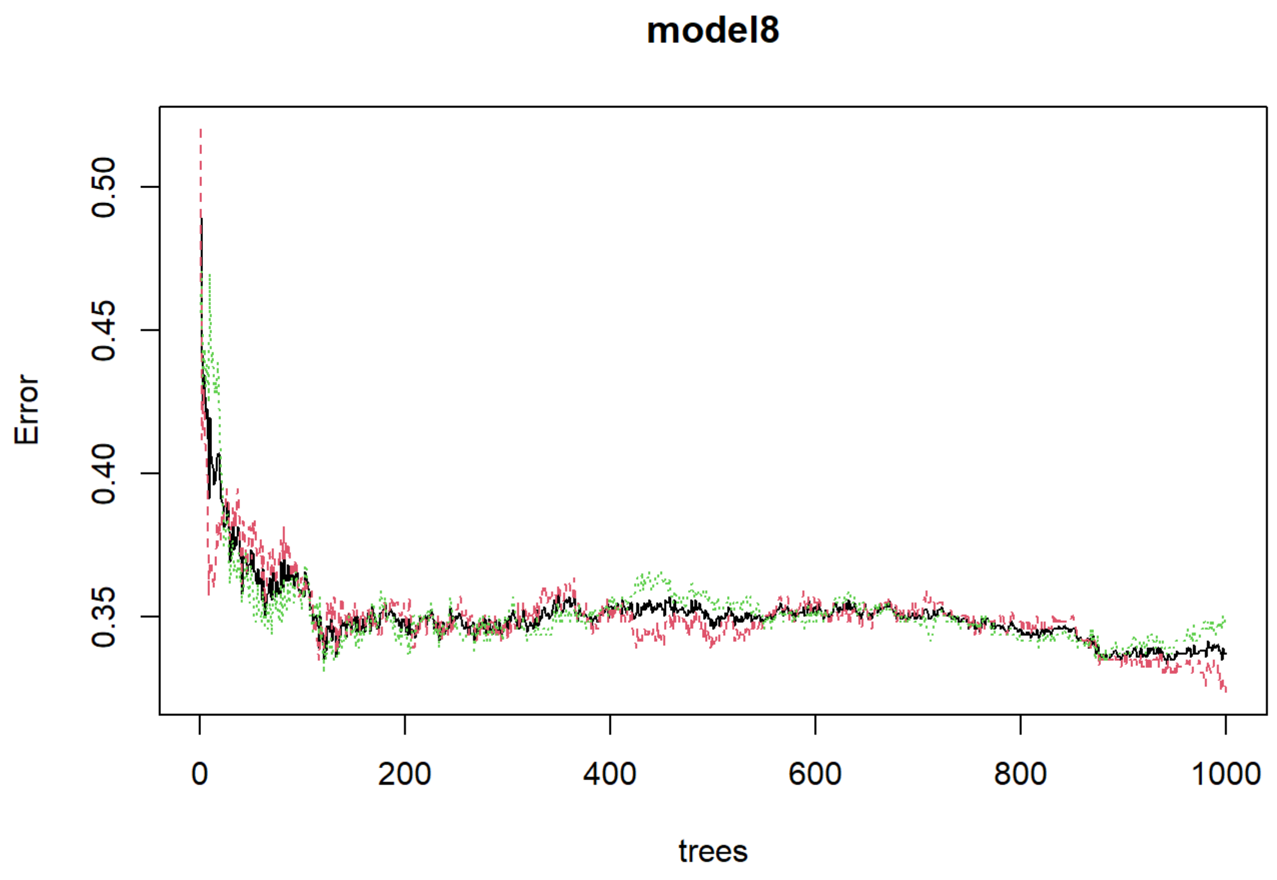


Let's choose an optimum value of ntree by creating another Random Forest model using under sampling dataset, but this time changing the value of ntree to 1000.

```
set.seed(100)
model8<-randomForest(y~.,data=under,ntree=1000,mtry=3)
model8
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = under, ntree = 1000, mtry = 3)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 33.7%
## Confusion matrix:
##     N   Y class.error
## N 305 146   0.3237251
## Y 158 293   0.3503326
```

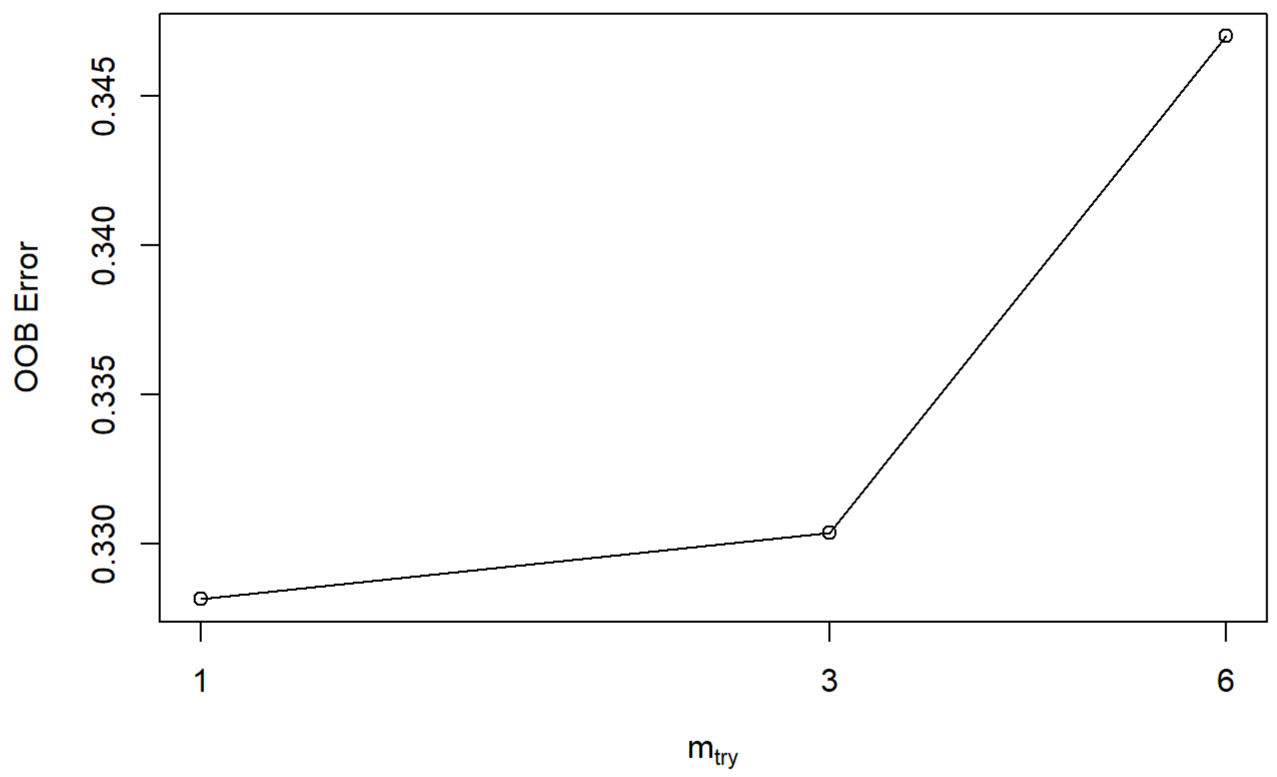
```
plot(model8)
```



The ntree value is changed to 1000 but still the OOB error estimate doesnot saturate at all but it comes down a bit. We will try to minimise this error estimate by varying the mtry value.

```
tuneRF(under[,-1],under[,1],stepFactor = 0.5,plot = TRUE,ntreeTry = 1000,trace = TRUE,improve
= 0.05)
```

```
## mtry = 3  OOB error = 33.04%
## Searching left ...
## mtry = 6     OOB error = 34.7%
## -0.05033557 0.05
## Searching right ...
## mtry = 1     OOB error = 32.82%
## 0.006711409 0.05
```



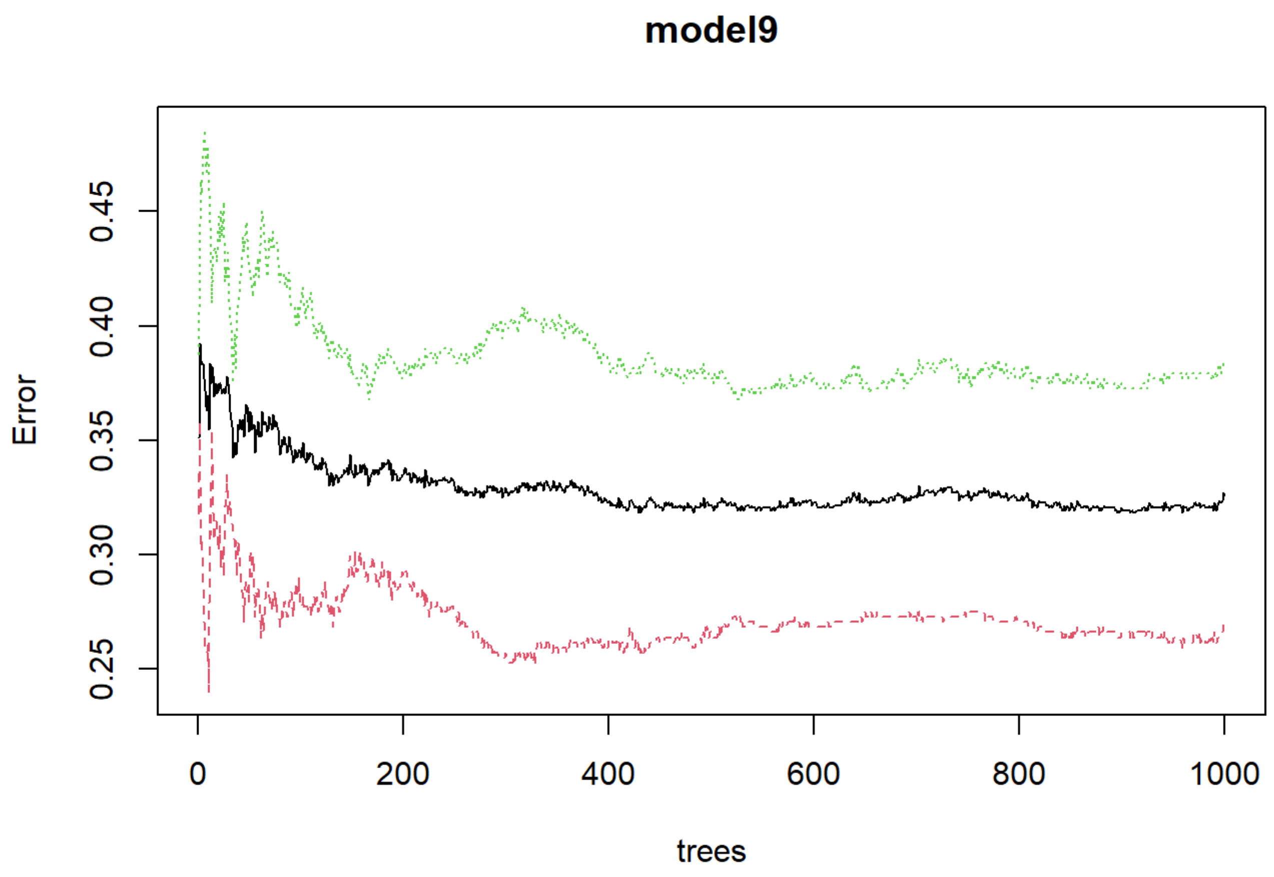
```
##       mtry  OOBError
## 1.OOB    1 0.3281596
## 3.OOB    3 0.3303769
## 6.OOB    6 0.3470067
```

As we can observe, that even after varying the values of ntree and mtry, the OOB error estimate couldn't be reduced much. Hence, we are building a new model with ntree=1000 and mtry=1.

```
set.seed(100)
model9<-randomForest(y~.,data = under,ntree=1000,mtry=1)
model9
```

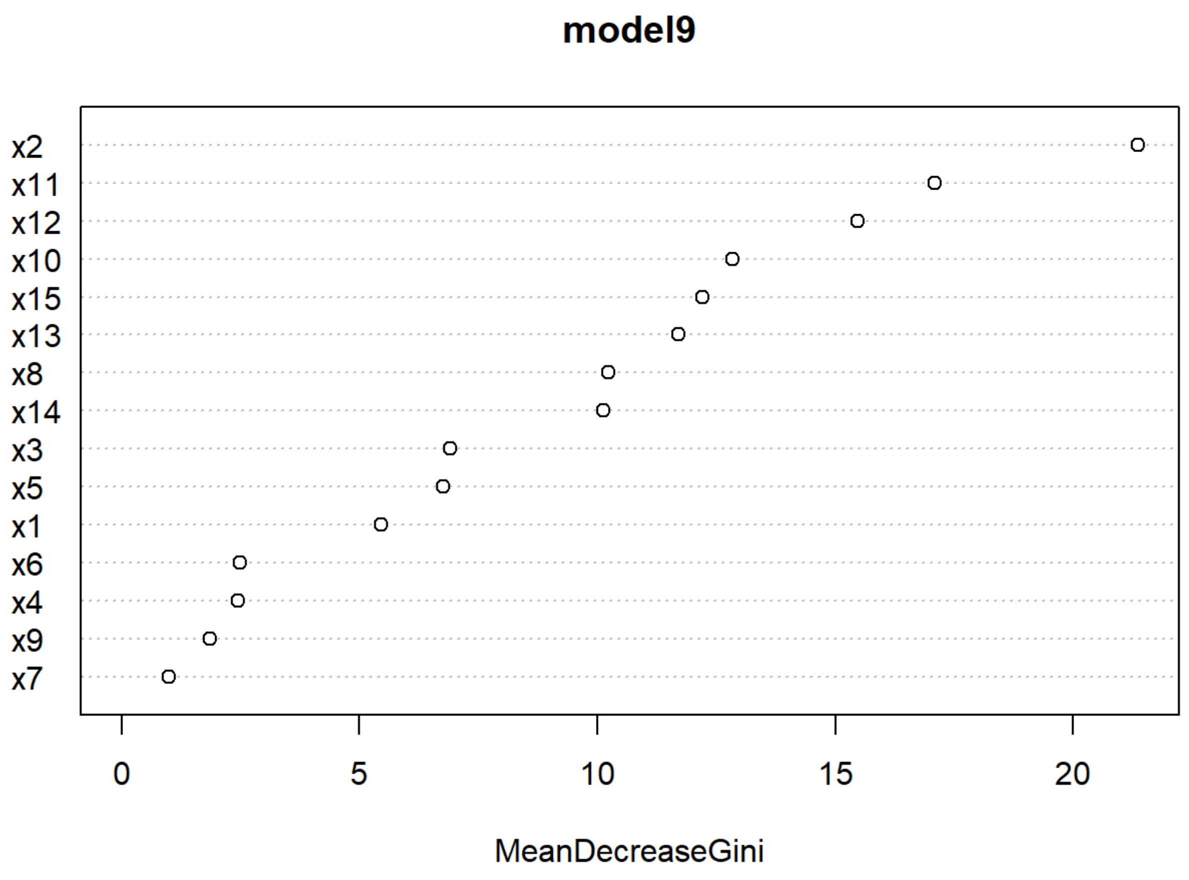
```
## 
## Call:
##  randomForest(formula = y ~ ., data = under, ntree = 1000, mtry = 1) 
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 1
## 
##         OOB estimate of  error rate: 32.59%
## Confusion matrix:
##     N   Y class.error
## N 329 122   0.2705100
## Y 172 279   0.3813747
```

```
plot(model9)
```



The OOB error estimate looks a bit better after tuning the parameters. We can have a look about the important features in our predictive model.

```
#Variable importance
varImpPlot(model9,sort = TRUE) #this graph measures how pure the nodes are at the end of the
 tree without each variable
```



```
importance(model9)
```

```
##     MeanDecreaseGini
## x1         5.4472184
## x2        21.3665006
## x3         6.9049430
## x4         2.4451471
## x5         6.7648279
## x6         2.4824118
## x7         0.9980211
## x8        10.2390019
## x9         1.8538435
## x10       12.8373561
## x11       17.0854827
## x12       15.4668349
## x13       11.7119534
## x14       10.1358505
## x15       12.1997296
```

```
varUsed(model9)
```

```
##  [1] 2698 4713 3822 2487 3696 1502 1130 2500 1661 4805 4625 4684 4718 4804 4774
```

Now, let's move on to performing predictions and creating the confusion matrix.

```
#prediction using test data
testpredrf_under<-predict(model9,test)
length(testpredrf_under)
```

```
## [1] 1271
```

```
#confusion matrix
ConfusionMatrix(testpredrf_under,test$y)
```

```
##       y_pred
## y_true   N   Y
##      N 743 335
##      Y  82 111
```

```
acc_rf_under=Accuracy(testpredrf_under,test$y)*100
prec_rf_under=Precision(test$y,testpredrf_under,positive = "Y")*100
rec_rf_under=Recall(test$y,testpredrf_under,positive = "Y")*100
spec_rf_under=Specificity(test$y,testpredrf_under,positive = "Y")*100
f1_rf_under=F1_Score(test$y,testpredrf_under,positive = "Y")*100
```

The model fitting with under sampling dataset was a bit below average.

Let's move on to model fitting using Random Forest technique with both over and under sampling dataset.

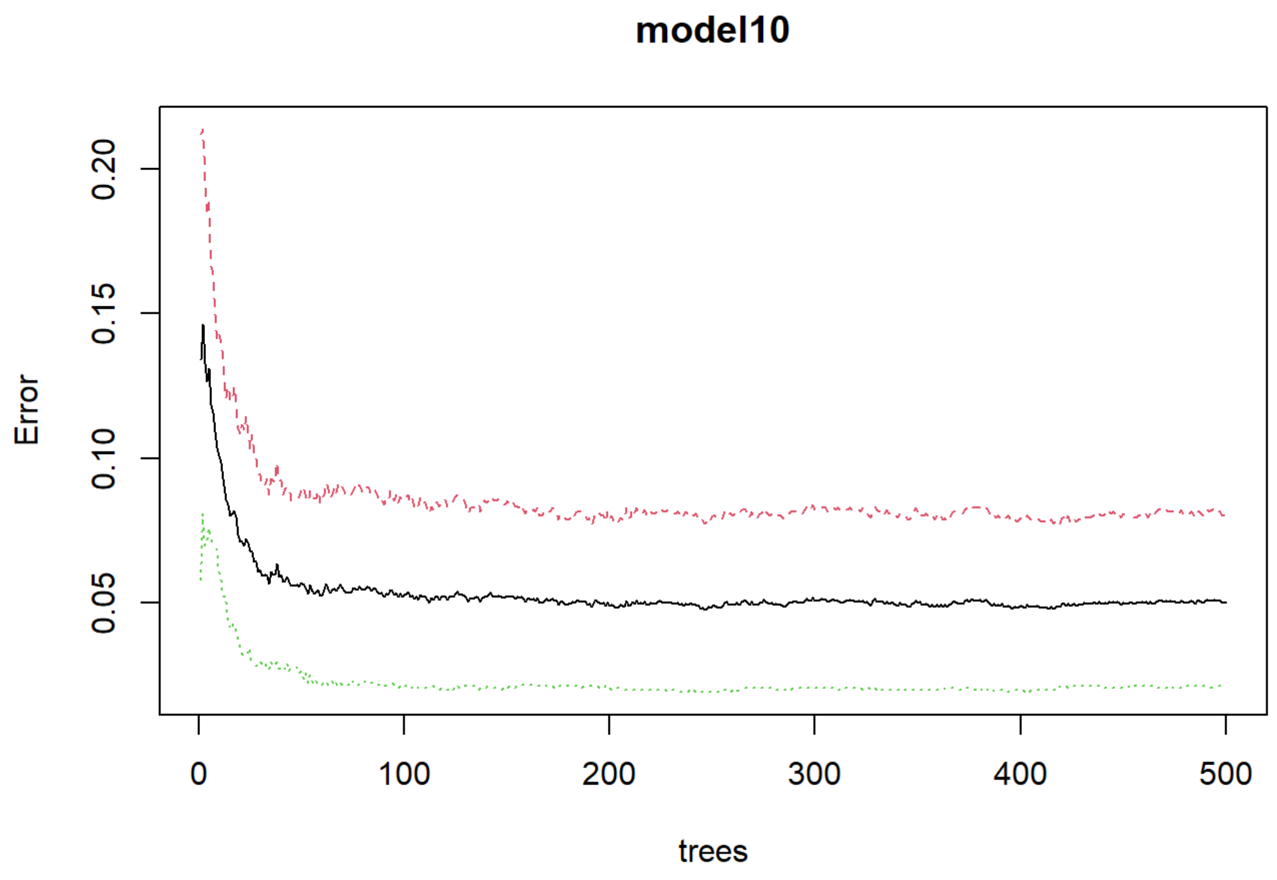
```
set.seed(100)
#building a primary RF model
model10<-randomForest(y~.,data=both)
model10
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = both)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 5.02%
## Confusion matrix:
##      N    Y class.error
## N 1334  117  0.08063405
## Y   32 1484  0.02110818
```

The OOB error estimate and the class errors indicates that our model fitting is very efficient and we need to tune our parameters to reduce the error estimates if possible.

Let's choose an optimum value of ntree.

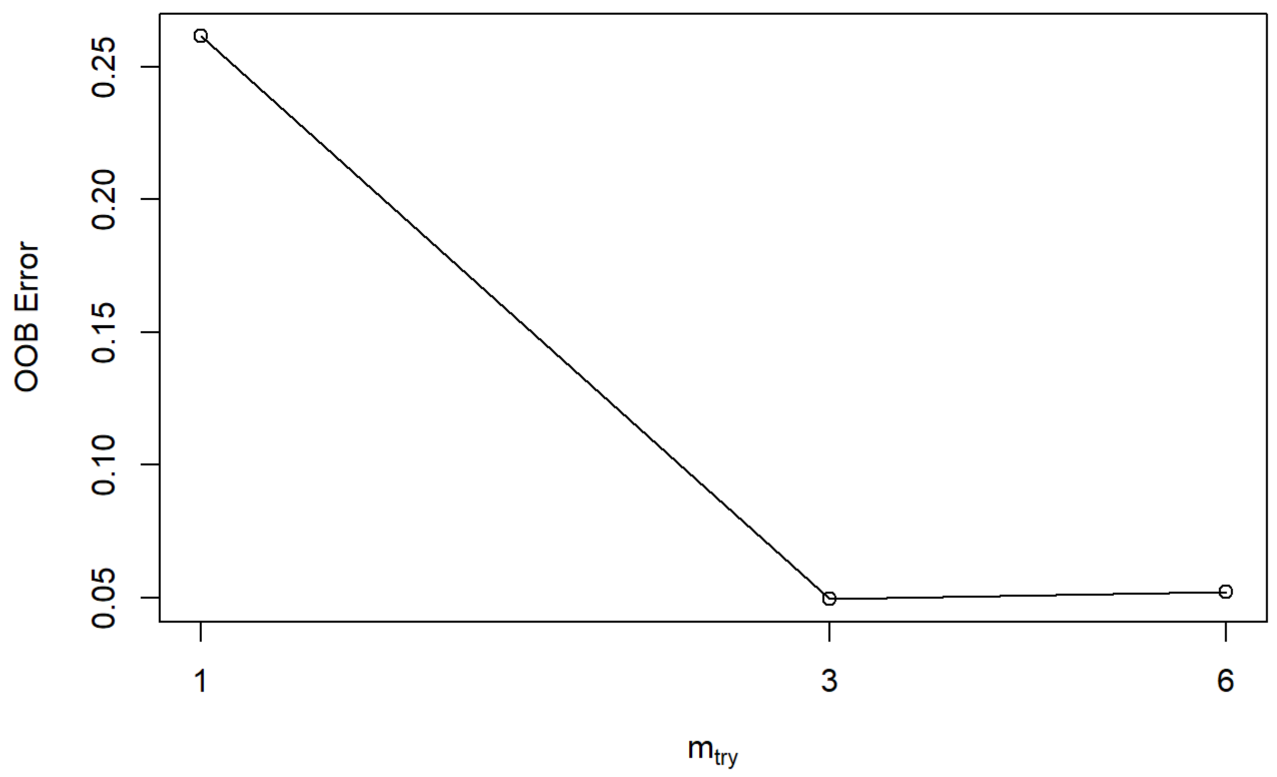
```
plot(model10)
```



The OOB error estimate gets saturated with 500 trees and thus, ntree=500 seems to be an optimum choice. Let's try to tune the mtry value if possible.

```
tuneRF(both[,-1],both[,1],stepFactor = 0.5,plot = TRUE,ntreeTry = 500,trace = TRUE,improve =
0.05)
```

```
## mtry = 3  OOB error = 4.95%
## Searching left ...
## mtry = 6     OOB error = 5.22%
## -0.05442177 0.05
## Searching right ...
## mtry = 1     OOB error = 26.15%
## -4.278912 0.05
```



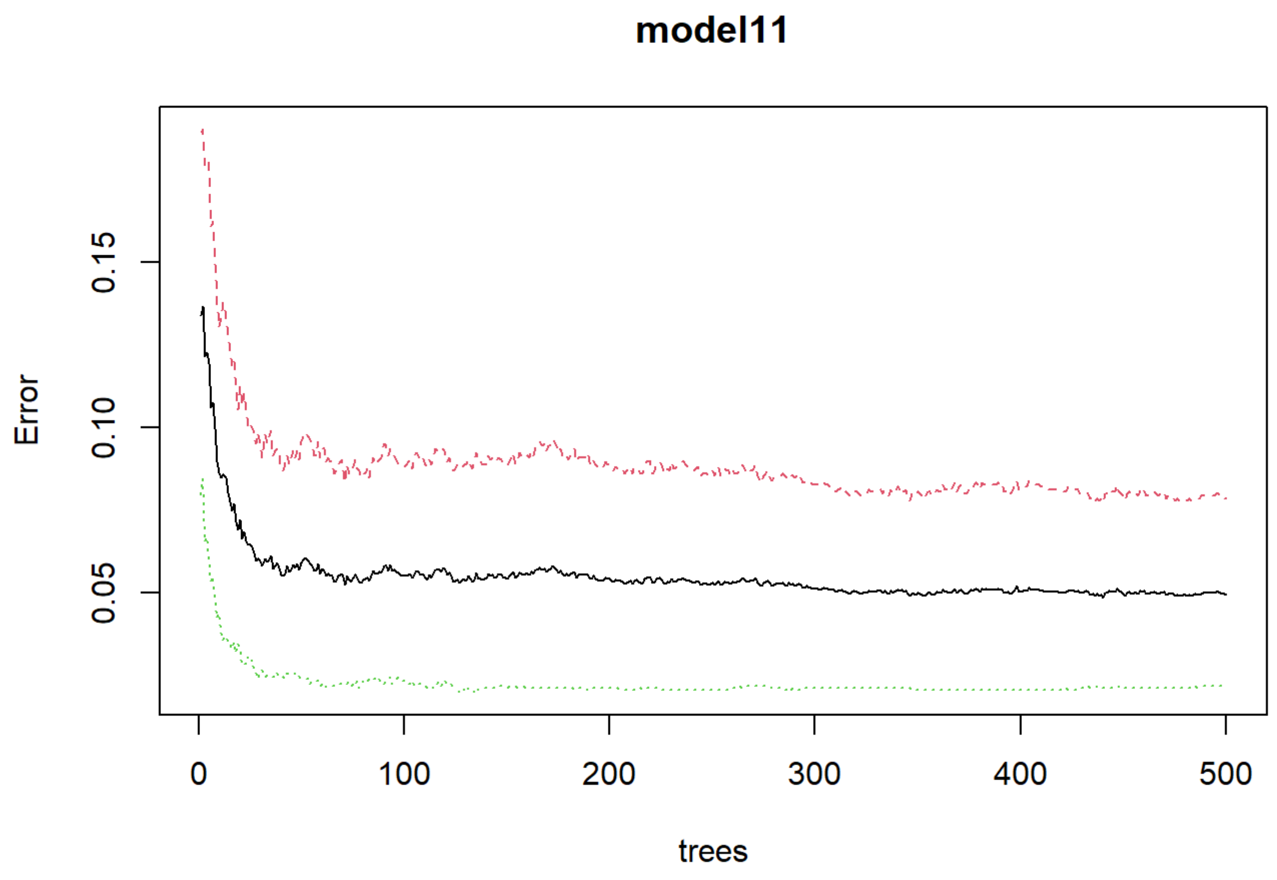
```
##       mtry   OOBError
## 1.OOB    1 0.26154365
## 3.OOB    3 0.04954499
## 6.OOB    6 0.05224132
```

The value of mtry seems to be perfect for 3, as the value of OOB error estimate comes down. But on further experimentation with the value of mtry, we found out that the OOB error estimate comes down even mor for mtry=4. Hence, we tune the model parameters accordingly.

```
set.seed(100)
model11<-randomForest(y~.,data = both,ntree=500,mtry=4)
model11
```

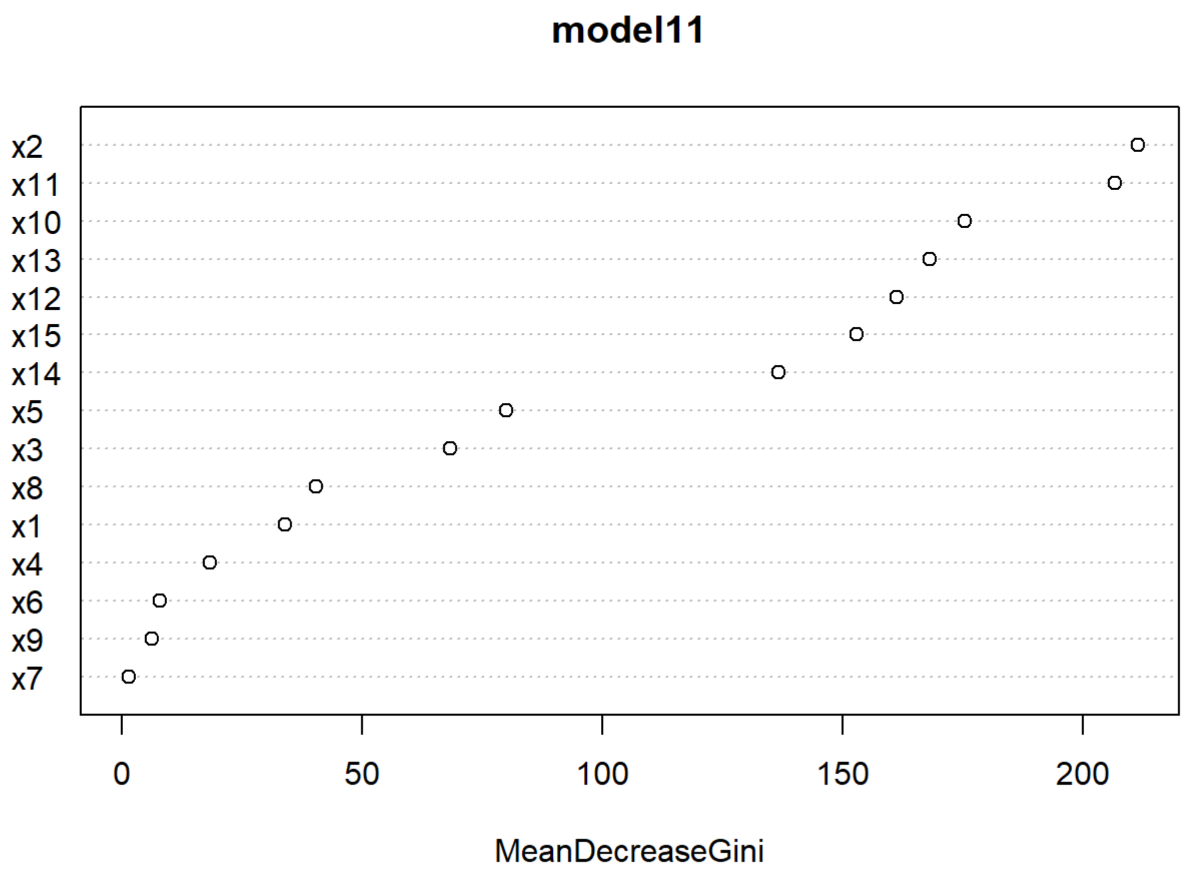
```
## 
## Call:
##  randomForest(formula = y ~ ., data = both, ntree = 500, mtry = 4) 
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
## 
##         OOB estimate of  error rate: 4.92%
## Confusion matrix:
##      N    Y class.error
## N 1337  114  0.07856651
## Y   32 1484  0.02110818
```

```
plot(model11)
```



We can have a look about the important features in our predictive model.

```
#Variable importance
varImpPlot(model11,sort = TRUE) #this graph measures how pure the nodes are at the end of the
tree without each variable
```



```
importance(model11)
```

```
##     MeanDecreaseGini
## x1         34.038569
## x2        211.582393
## x3         68.350876
## x4         18.472801
## x5         79.969344
## x6          7.927319
## x7          1.503283
## x8         40.448927
## x9          6.206941
## x10       175.586312
## x11       206.806055
## x12       161.301042
## x13       168.098204
## x14       136.777625
## x15       153.082238
```

```
varUsed(model11)
```

```
##  [1]  3784 19951  9784  2780 11291  1050   286  2203   983 22471 22079 20819
## [13] 22869 19629 21068
```

Let's move on to our predictions using the testing datset and evaluate our model using confusion matrix.

```
#prediction using test data
testpredrf_both<-predict(model11,test)
length(testpredrf_both)
```

```
## [1] 1271
```

```
#confusion matrix
ConfusionMatrix(testpredrf_both,test$y)
```

```
##       y_pred
## y_true   N   Y
##      N 934 144
##      Y 129  64
```

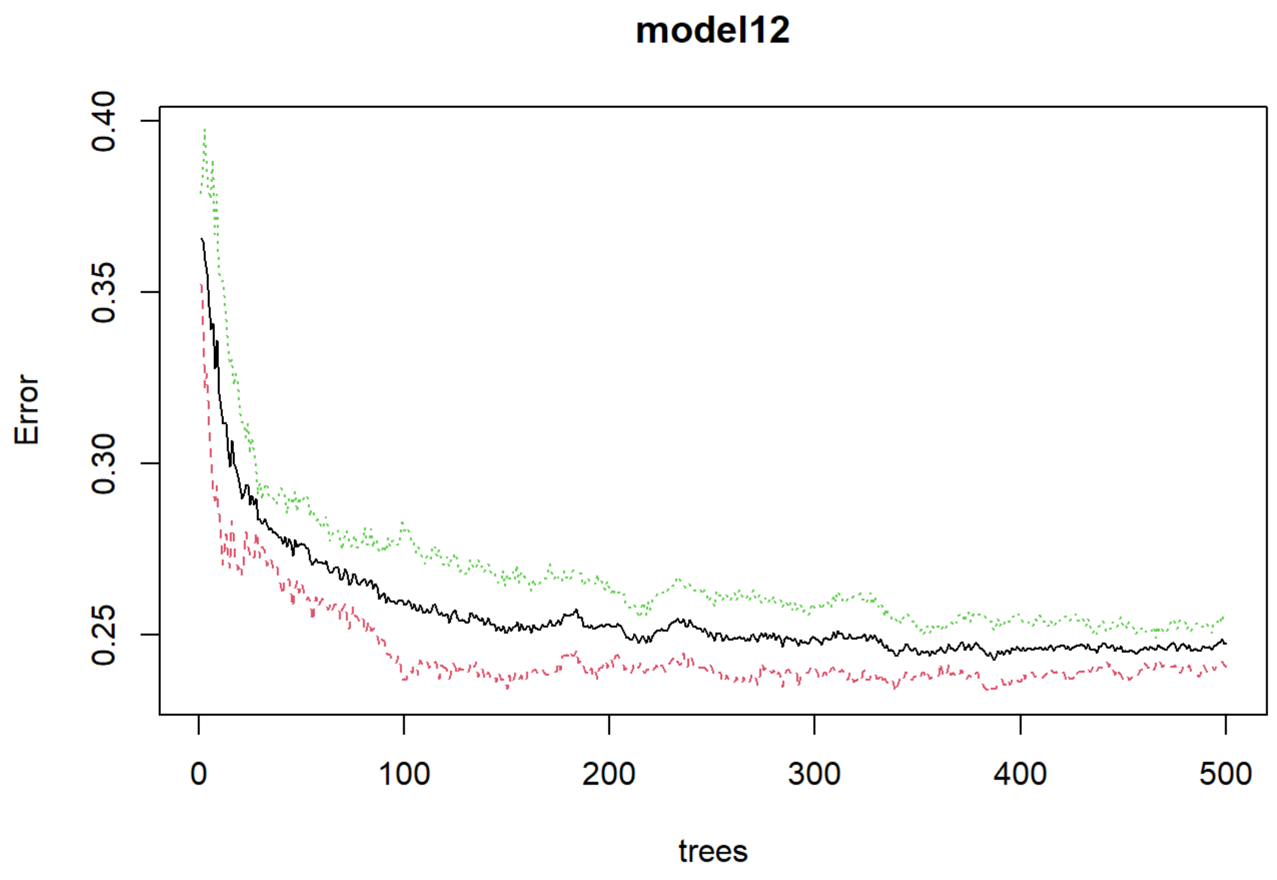
```
acc_rf_both=Accuracy(testpredrf_both,test$y)*100
prec_rf_both=Precision(test$y,testpredrf_both,positive = "Y")*100
rec_rf_both=Recall(test$y,testpredrf_both,positive = "Y")*100
spec_rf_both=Specificity(test$y,testpredrf_both,positive = "Y")*100
f1_rf_both=F1_Score(test$y,testpredrf_both,positive = "Y")*100
```

Let's move on to model fitting using Random Forest technique with synthetic dataset.

```
set.seed(100)
#building a primary RF model
model12<-randomForest(y~.,data=syndata)
model12
```

```
## 
## Call:
##  randomForest(formula = y ~ ., data = syndata)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
## 
##         OOB estimate of  error rate: 24.77%
## Confusion matrix:
##      N    Y class.error
## N 1526  484   0.2407960
## Y  507 1483   0.2547739
```

```
plot(model12)
```



The OOB error estimate and the class errors indicates that our model fitting is not very efficient and we need to tune our parameters to reduce the error estimates.

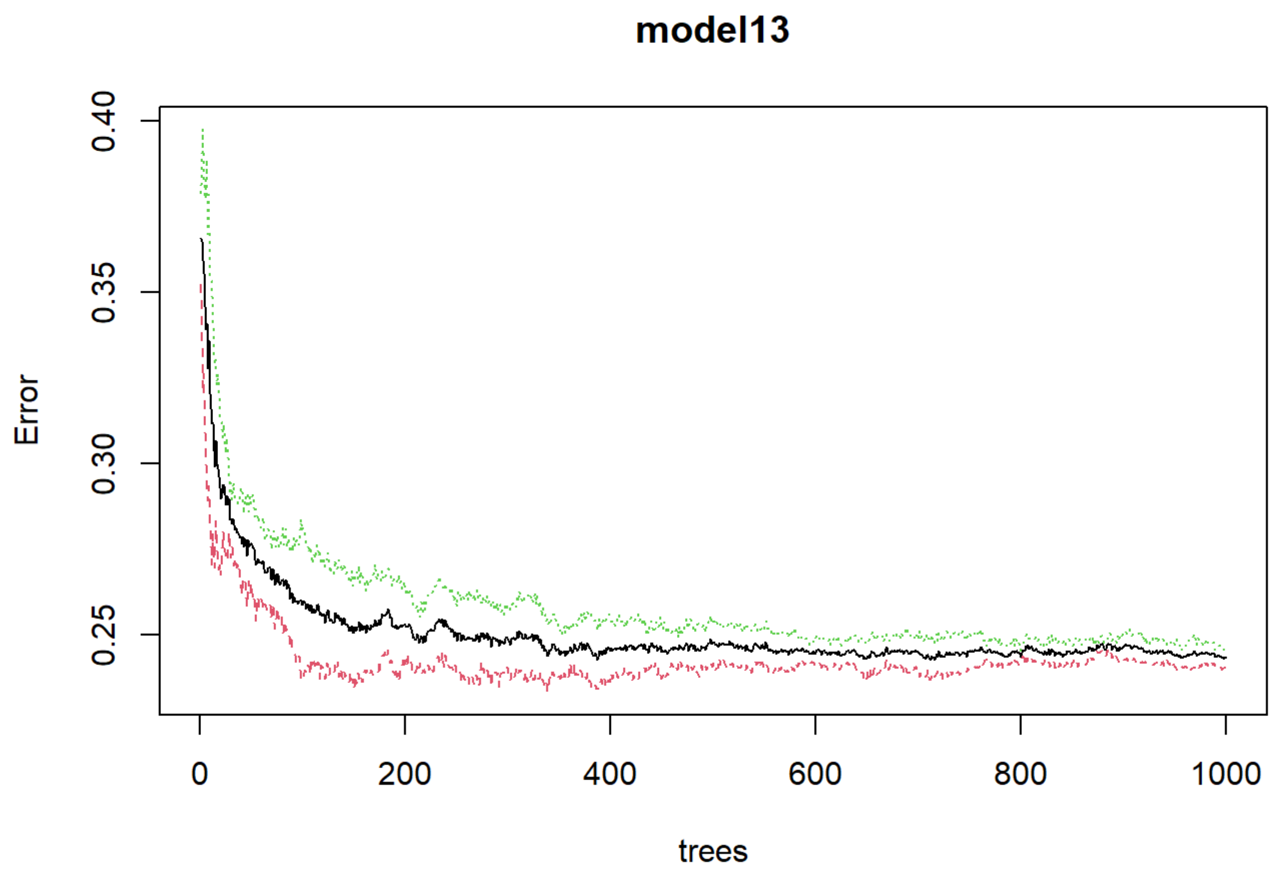
Let's choose an optimum value of ntree by creating another Random Forest model using under sampling dataset, but this time changing the value of ntree to 1000.

```
set.seed(100)
model13<-randomForest(y~.,data=syndata,ntree=1000,mtry=3)
model13
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = syndata, ntree = 1000, mtry = 3)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 24.32%
## Confusion matrix:
##      N    Y class.error
## N 1525  485   0.2412935
## Y  488 1502   0.2452261
```

The OOB error estimate comes down a bit when the value of ntree is fixed to 1000.

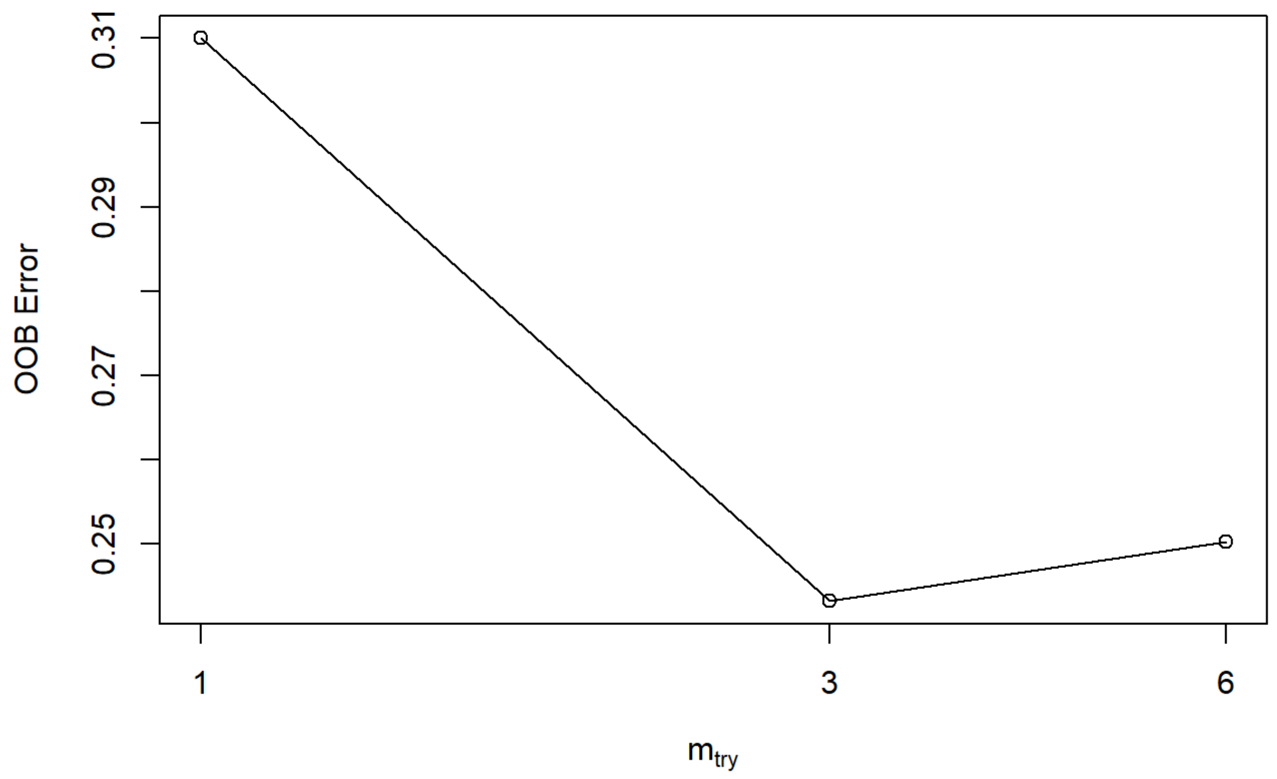
```
plot(model13)
```



Let’s try to tune the mtry value if possible.

```
tuneRF(syndata[,-1],syndata[,1],stepFactor = 0.5,plot = TRUE,ntreeTry = 1000,trace = TRUE,imp
rove = 0.05)
```

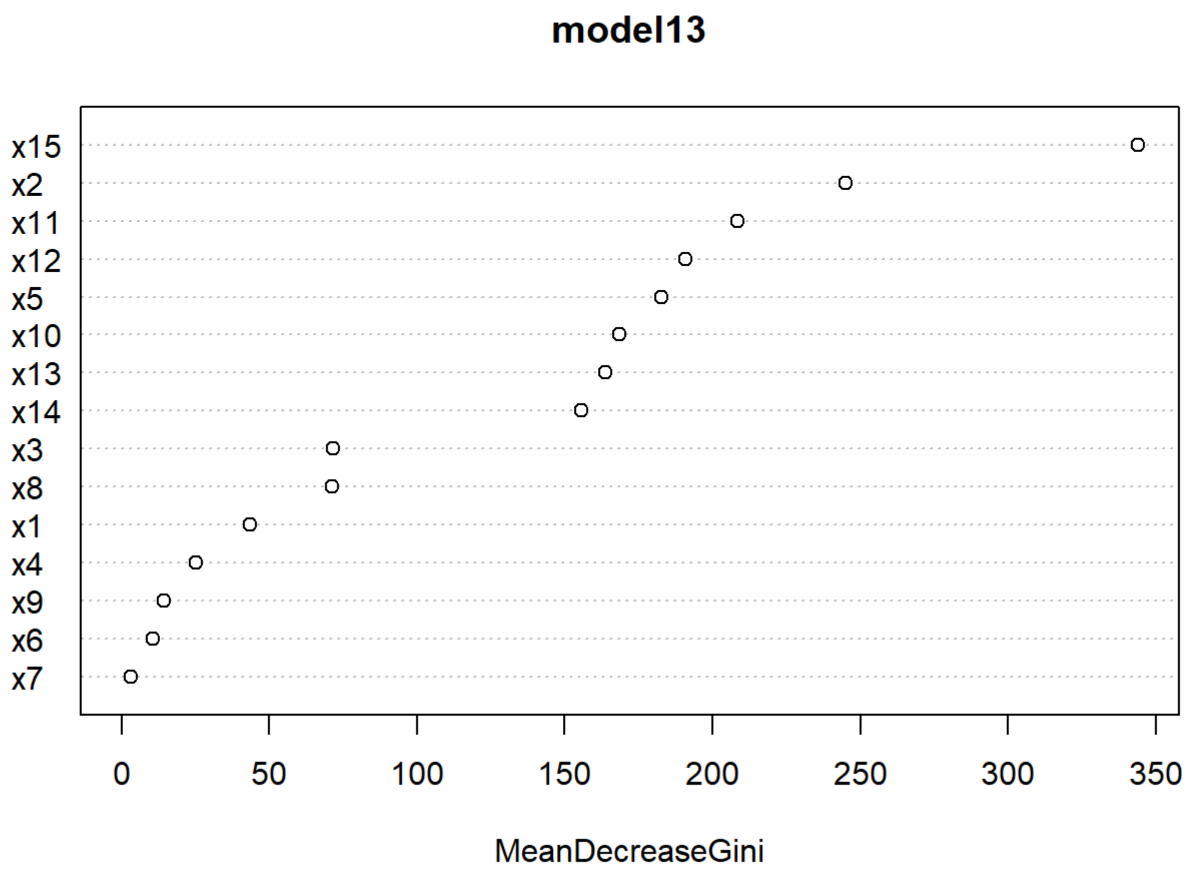
```
## mtry = 3  OOB error = 24.32%
## Searching left ...
## mtry = 6     OOB error = 25.02%
## -0.02877698 0.05
## Searching right ...
## mtry = 1     OOB error = 31%
## -0.274409 0.05
```



```
##       mtry OOBError
## 1.OOB    1  0.31000
## 3.OOB    3  0.24325
## 6.OOB    6  0.25025
```

mtry=3 seems to be an appropriate choice. Hence, model13 is an appropriate model for synthetic dataset. We can have a look about the important features in our predictive model.

```
#Variable importance
varImpPlot(model13,sort = TRUE) #this graph measures how pure the nodes are at the end of the
tree without each variable
```



```
importance(model13)
```

```
##     MeanDecreaseGini
## x1         43.508330
## x2        245.166291
## x3         71.706144
## x4         25.149444
## x5        182.642340
## x6         10.483515
## x7          3.242731
## x8         71.096260
## x9         14.314167
## x10       168.521174
## x11       208.415902
## x12       190.877697
## x13       163.701818
## x14       155.677152
## x15       344.056783
```

```
varUsed(model13)
```

```
##  [1] 13012 74146 31946 15292 73394  4761  1677  8295  3940 72155 73362 73442
## [13] 71792 70798 78765
```

Let's move on to our predictions using the testing datset and evaluate our model using confusion matrix.

```
#prediction using test data
testpredrf_syn<-predict(model13,test)
length(testpredrf_syn)
```

```
## [1] 1271
```

```
#confusion matrix
ConfusionMatrix(testpredrf_syn,test$y)
```

```
##       y_pred
## y_true   N   Y
##      N 877 201
##      Y 115  78
```

```
acc_rf_syn=Accuracy(testpredrf_syn,test$y)*100
prec_rf_syn=Precision(test$y,testpredrf_syn,positive = "Y")*100
rec_rf_syn=Recall(test$y,testpredrf_syn,positive = "Y")*100
spec_rf_syn=Specificity(test$y,testpredrf_syn,positive = "Y")*100
f1_rf_syn=F1_Score(test$y,testpredrf_syn,positive = "Y")*100
```

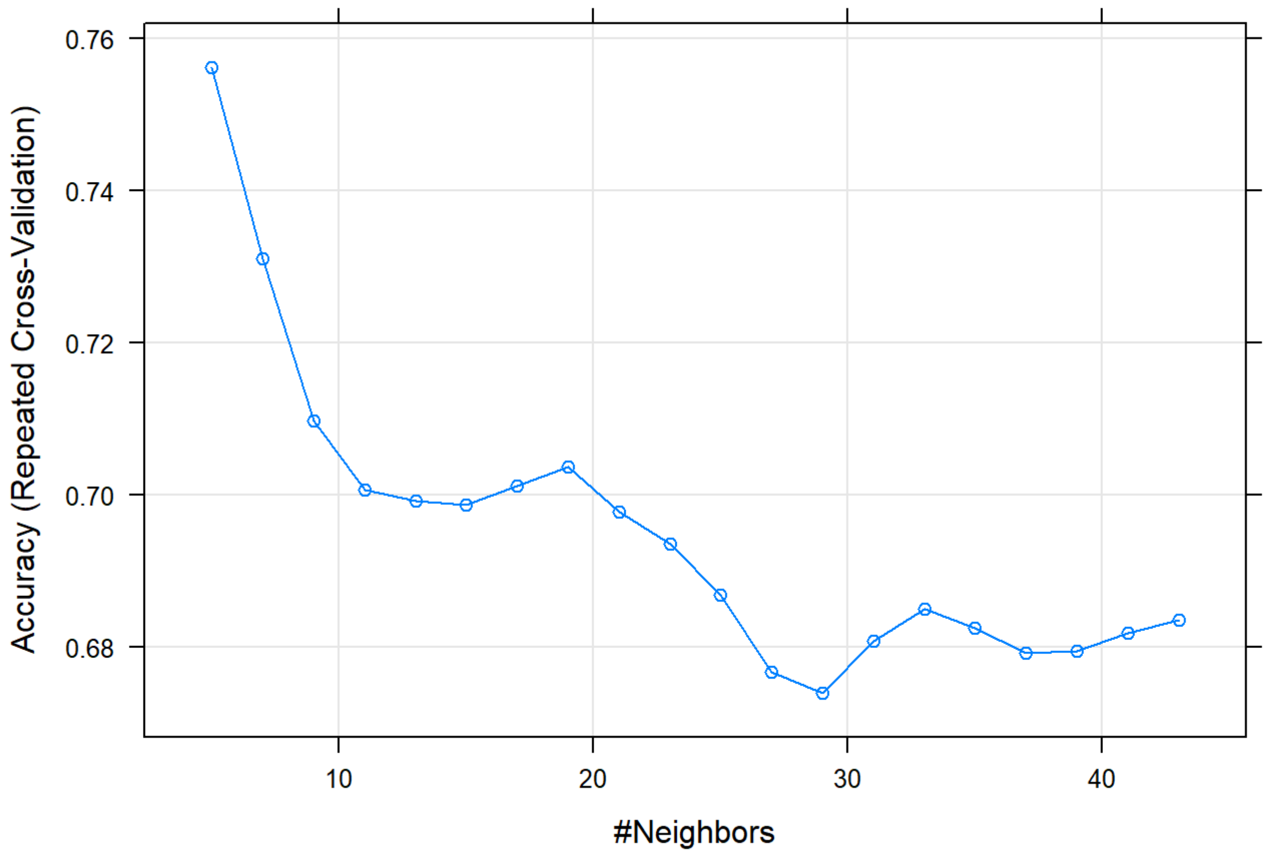
Now after building our predictive models using Logistic Regression and Random Forest, we move on to building the model using K Nearest Neighbors.

Let's build a KNN model using the over sampling dataset.

```
trControl <- trainControl(method = "repeatedcv", #repeated cross-validation
                          number = 10,  # number of resampling iterations
                          repeats = 3) #,  # sets of folds to for repeated cross-validation
#classProbs = TRUE, summaryFunction = twoClassSummary)  # classProbs needed for ROC
set.seed(1234)
fit_over <- train(y~ .,
                   data = over,
                   method = "knn",
                   tuneLength = 20,
                   trControl = trControl,
                   preProc = c("center", "scale"))  # necessary task
#model performance
fit_over
```

```
## k-Nearest Neighbors
##
## 5032 samples
##   15 predictor
##    2 classes: 'N', 'Y'
##
## Pre-processing: centered (17), scaled (17)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 4528, 4529, 4528, 4529, 4528, 4529, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.7562251  0.5124460
##    7  0.7310566  0.4621147
##    9  0.7097342  0.4194721
##   11  0.7007172  0.4014461
##   13  0.6991987  0.3984075
##   15  0.6987362  0.3974871
##   17  0.7011888  0.4023890
##   19  0.7036377  0.4072854
##   21  0.6978063  0.3956206
##   23  0.6935043  0.3870132
##   25  0.6868796  0.3737663
##   27  0.6766763  0.3533601
##   29  0.6738967  0.3478001
##   31  0.6807837  0.3615742
##   33  0.6850252  0.3700569
##   35  0.6824374  0.3648817
##   37  0.6792584  0.3585239
##   39  0.6795249  0.3590600
##   41  0.6818392  0.3636884
##   43  0.6835646  0.3671379
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

```
plot(fit_over)
```



Now, we perform the predictions and create the confusion matrix for evaluation of our model

```
testpredknn_over <- predict(fit_over, newdata = test )
length(testpredknn_over)
```

```
## [1] 1271
```

```
#confusion matrix
ConfusionMatrix(testpredknn_over,test$y)
```

```
##         y_pred
## y_true    N   Y
##        N 606 472
##        Y  76 117
```

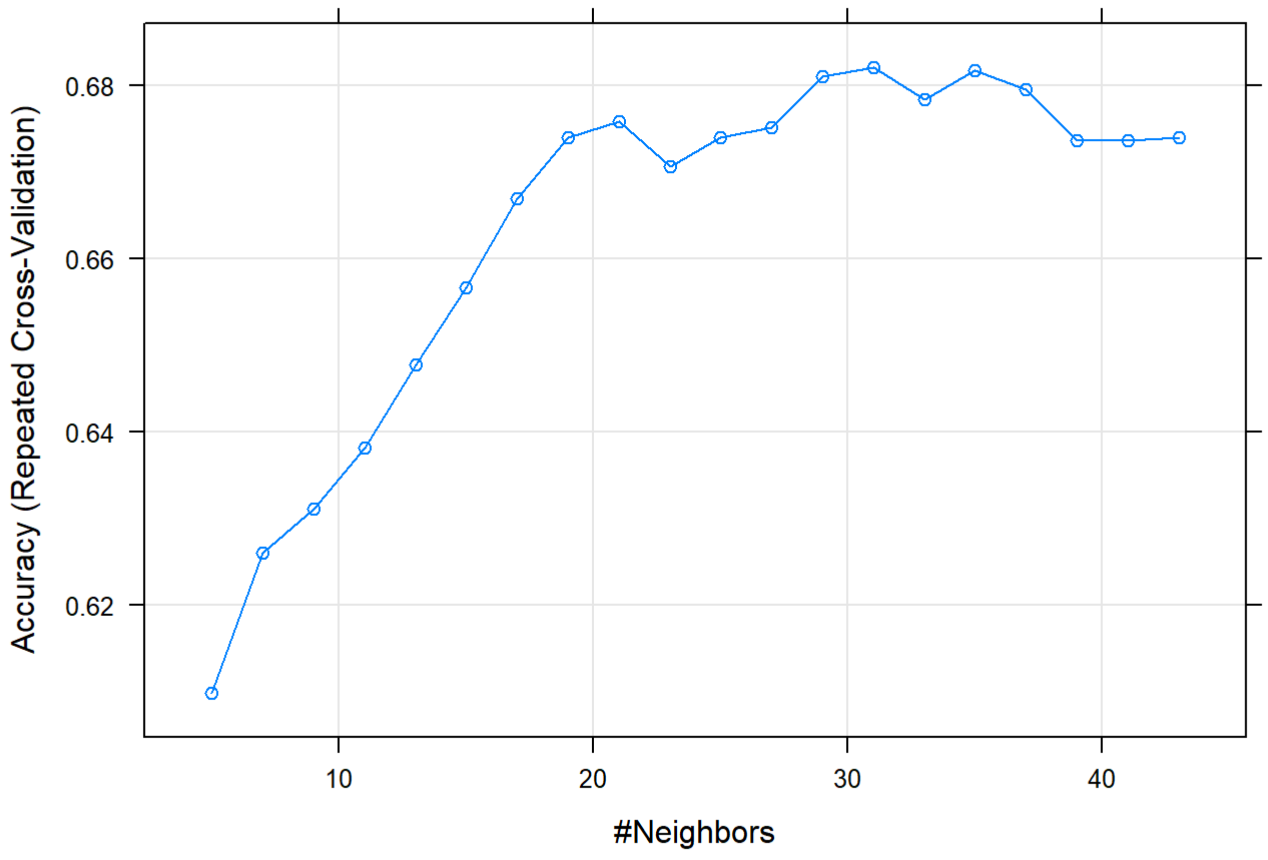
```
acc_knn_over=Accuracy(testpredknn_over,test$y)*100
prec_knn_over=Precision(test$y,testpredknn_over,positive = "Y")*100
rec_knn_over=Recall(test$y,testpredknn_over,positive = "Y")*100
spec_knn_over=Specificity(test$y,testpredknn_over,positive = "Y")*100
f1_knn_over=F1_Score(test$y,testpredknn_over,positive = "Y")*100
```

Now, let's build a KNN model using the under sampling dataset.

```
set.seed(1234)
fit_under <- train(y~ .,
                  data = under,
                  method = "knn",
                  tuneLength = 20,
                  trControl = trControl,
                  preProc = c("center", "scale"))  # necessary task
#model performance
fit_under
```

```
## k-Nearest Neighbors
##
## 902 samples
##  15 predictor
##   2 classes: 'N', 'Y'
##
## Pre-processing: centered (17), scaled (17)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 812, 812, 811, 812, 812, 812, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.6097651  0.2194627
##    7  0.6259884  0.2518891
##    9  0.6311249  0.2621441
##   11  0.6381542  0.2762254
##   13  0.6477715  0.2954968
##   15  0.6566201  0.3131788
##   17  0.6669782  0.3339055
##   19  0.6739949  0.3479487
##   21  0.6758264  0.3515957
##   23  0.6706776  0.3413008
##   25  0.6739664  0.3478660
##   27  0.6751099  0.3501639
##   29  0.6810440  0.3620198
##   31  0.6821187  0.3641641
##   33  0.6784191  0.3567695
##   35  0.6817605  0.3634452
##   37  0.6795381  0.3590032
##   39  0.6736204  0.3471728
##   41  0.6736407  0.3472419
##   43  0.6740151  0.3479989
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 31.
```

```
plot(fit_under)
```



Now, we perform the predictions and create the confusion matrix for evaluation of our model

```
testpredknn_under <- predict(fit_under, newdata = test )
length(testpredknn_under)
```

```
## [1] 1271
```

```
#confusion matrix
ConfusionMatrix(testpredknn_under,test$y)
```

```
##         y_pred
## y_true   N   Y
##        N 721 357
##        Y  77 116
```

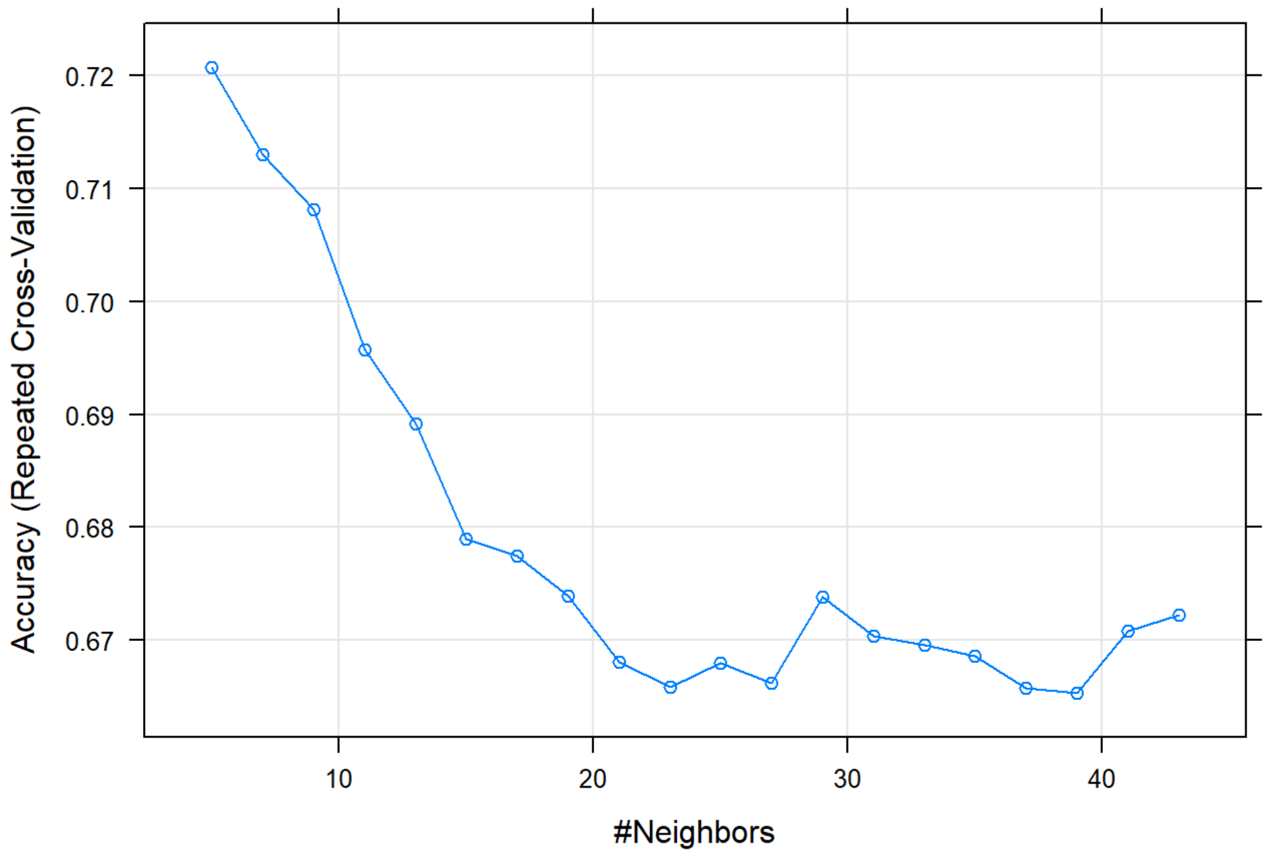
```
acc_knn_under=Accuracy(testpredknn_under,test$y)*100
prec_knn_under=Precision(test$y,testpredknn_under,positive = "Y")*100
rec_knn_under=Recall(test$y,testpredknn_under,positive = "Y")*100
spec_knn_under=Specificity(test$y,testpredknn_under,positive = "Y")*100
f1_knn_under=F1_Score(test$y,testpredknn_under,positive = "Y")*100
```

Now, let's build a KNN model using the both over and under sampling dataset.

```
set.seed(1234)
fit_both <- train(y~ .,
                   data = both,
                   method = "knn",
                   tuneLength = 20,
                   trControl = trControl,
                   preProc = c("center", "scale"))  # necessary task
#model performance
fit_both
```

```
## k-Nearest Neighbors
##
## 2967 samples
##   15 predictor
##    2 classes: 'N', 'Y'
##
## Pre-processing: centered (17), scaled (17)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 2670, 2671, 2669, 2670, 2670, 2671, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.7207085  0.4383163
##    7  0.7129534  0.4232996
##    9  0.7081145  0.4137095
##   11  0.6957556  0.3889383
##   13  0.6891247  0.3757722
##   15  0.6790145  0.3557742
##   17  0.6774463  0.3528450
##   19  0.6739682  0.3460498
##   21  0.6681259  0.3344763
##   23  0.6658726  0.3299265
##   25  0.6680167  0.3342038
##   27  0.6662203  0.3306268
##   29  0.6738578  0.3459088
##   31  0.6703714  0.3390605
##   33  0.6695824  0.3376024
##   35  0.6685791  0.3357732
##   37  0.6657664  0.3302671
##   39  0.6653152  0.3293705
##   41  0.6708215  0.3404101
##   43  0.6722809  0.3433912
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

```
plot(fit_both)
```



Now, we perform the predictions and create the confusion matrix for evaluation of our model

```
testpredknn_both <- predict(fit_both, newdata = test )
length(testpredknn_both)
```

```
## [1] 1271
```

```
#confusion matrix
ConfusionMatrix(testpredknn_both,test$y)
```

```
##         y_pred
## y_true    N   Y
##        N 561 517
##        Y  78 115
```

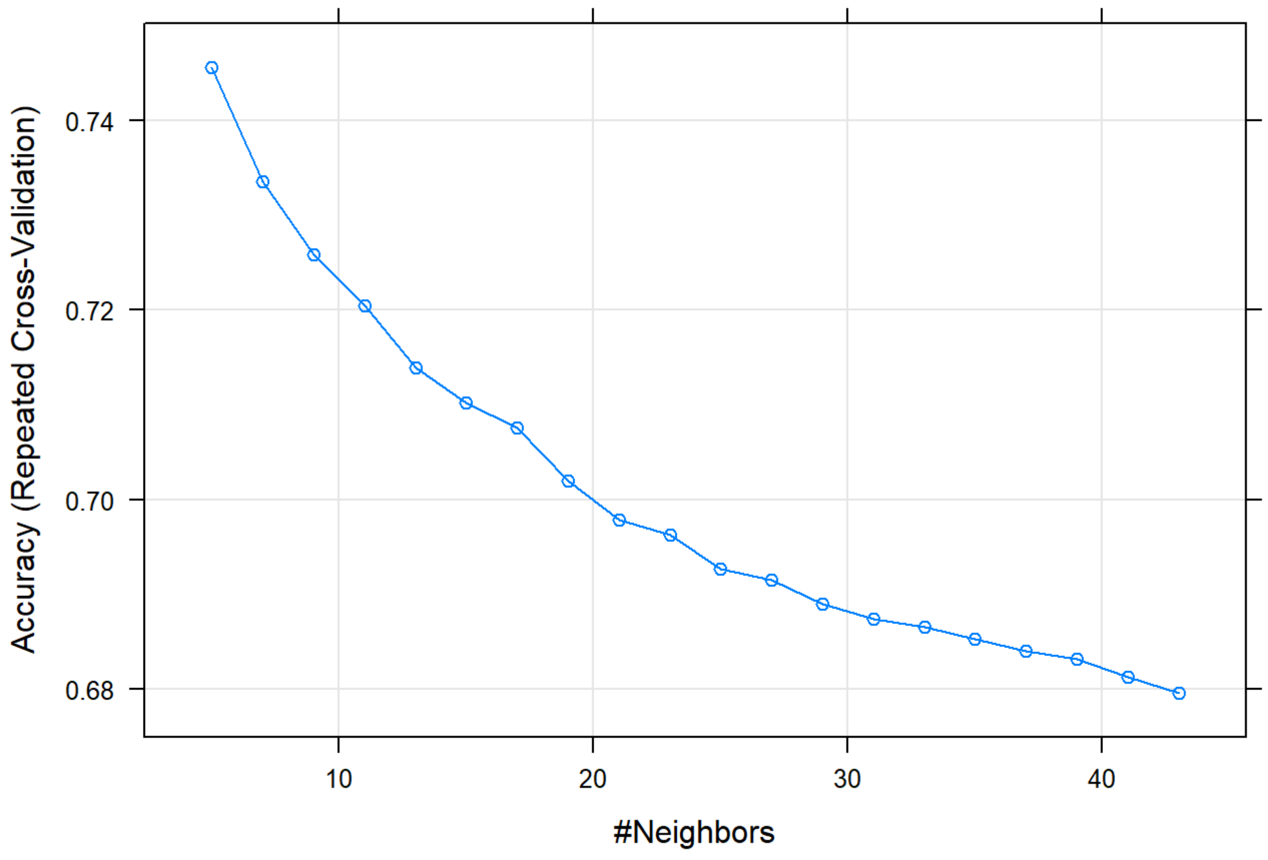
```
acc_knn_both=Accuracy(testpredknn_both,test$y)*100
prec_knn_both=Precision(test$y,testpredknn_both,positive = "Y")*100
rec_knn_both=Recall(test$y,testpredknn_both,positive = "Y")*100
spec_knn_both=Specificity(test$y,testpredknn_both,positive = "Y")*100
f1_knn_both=F1_Score(test$y,testpredknn_both,positive = "Y")*100
```

Now, let's build an KNN model using the synthetic dataset.

```
set.seed(1234)
fit_synd <- train(y~ .,
                  data = syndata,
                  method = "knn",
                  tuneLength = 20,
                  trControl = trControl,
                  preProc = c("center", "scale"))  # necessary task
#model performance
fit_synd
```

```
## k-Nearest Neighbors
##
## 4000 samples
##   15 predictor
##    2 classes: 'N', 'Y'
##
## Pre-processing: centered (17), scaled (17)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 3600, 3600, 3600, 3600, 3600, 3600, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.7455833  0.4910222
##    7  0.7335000  0.4667884
##    9  0.7258333  0.4513904
##   11  0.7204167  0.4405252
##   13  0.7139167  0.4274779
##   15  0.7101667  0.4199405
##   17  0.7075833  0.4147407
##   19  0.7020000  0.4035247
##   21  0.6978333  0.3951630
##   23  0.6963333  0.3921347
##   25  0.6926667  0.3847744
##   27  0.6915000  0.3824284
##   29  0.6890000  0.3774061
##   31  0.6874167  0.3742316
##   33  0.6865833  0.3725487
##   35  0.6853333  0.3700372
##   37  0.6840833  0.3675124
##   39  0.6831667  0.3656904
##   41  0.6813333  0.3620039
##   43  0.6795833  0.3584919
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

```
plot(fit_synd)
```



Now, we perform the predictions and create the confusion matrix for evaluation of our model

```
testpredknn_syn <- predict(fit_synd, newdata = test )
length(testpredknn_syn)
```

```
## [1] 1271
```

```
#confusion matrix
ConfusionMatrix(testpredknn_syn,test$y)
```

```
##        y_pred
## y_true   N   Y
##      N 809 269
##      Y 123  70
```

```
acc_knn_syn=Accuracy(testpredknn_syn,test$y)*100
prec_knn_syn=Precision(test$y,testpredknn_syn,positive = "Y")*100
rec_knn_syn=Recall(test$y,testpredknn_syn,positive = "Y")*100
spec_knn_syn=Specificity(test$y,testpredknn_syn,positive = "Y")*100
f1_knn_syn=F1_Score(test$y,testpredknn_syn,positive = "Y")*100
```

Now, we move on to the next classifier and that is Naive Bayes Classifier. Let's build our classification model using Naive Bayes Classifier with over sampling dataset.

```
nb_overfit=naiveBayes(y~.,data=over)
nb_overfit
```

```
## 
## Naive Bayes Classifier for Discrete Predictors
## 
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
## 
## A-priori probabilities:
## Y
##   N   Y 
## 0.5 0.5 
## 
## Conditional probabilities:
##    x1
## Y           F         M
##   N 0.5914149 0.4085851
##   Y 0.4558824 0.5441176
## 
##    x2
## Y       [,1]    [,2]
##   N 48.83824 8.41104
##   Y 54.57552 7.90044
## 
##    x3
## Y           1         2         3         4
##   N 0.3855326 0.3310811 0.1697138 0.1136725
##   Y 0.5035771 0.2301272 0.1486486 0.1176471
## 
##    x4
## Y           N         Y
##   N 0.5091415 0.4908585
##   Y 0.4876789 0.5123211
## 
##    x5
## Y        [,1]     [,2]
##   N  8.804871 11.72025
##   Y 10.575919 13.02184
## 
##    x6
## Y            N          Y
##   N 0.97774245 0.02225755
##   Y 0.93680445 0.06319555
## 
##    x7
## Y             N           Y
##   N 0.996025437 0.003974563
##   Y 0.979729730 0.020270270
## 
##    x8
## Y           N         Y
##   N 0.7229730 0.2770270
##   Y 0.4900636 0.5099364
## 
##    x9
## Y            N          Y
##   N 0.98171701 0.01828299
##   Y 0.93640700 0.06359300
## 
##    x10
## Y       [,1]     [,2]
##   N 234.1301 43.65478
##   Y 242.9065 43.70456
## 
##    x11
## Y       [,1]     [,2]
##   N 130.3120 20.47253
##   Y 142.8281 25.03786
## 
##    x12
## Y       [,1]     [,2]
##   N 82.07711 11.36465
##   Y 86.74523 13.05569
## 
##    x13
## Y       [,1]     [,2]
##   N 25.65359 3.972571
##   Y 26.53113 4.155533
## 
##    x14
## Y       [,1]     [,2]
##   N 75.53776 11.86765
##   Y 76.83005 12.26639
## 
##    x15
## Y       [,1]     [,2]
##   N 80.77473 16.85909
##   Y 90.09618 41.44160
```

```
summary(nb_overfit)
```

```
##           Length Class  Mode
## apriori    2     table  numeric
## tables    15     -none- list
## levels     2     -none- character
## isnumeric 15     -none- logical
## call       4     -none- call
```

Now, we perform the predictions and create the confusion matrix for evaluation of our model

```
testpred nb_over=predict(nb_overfit,newdata = test)
length(testprednb_over)
```

```
## [1] 1271
```

```
#confusion matrix
ConfusionMatrix(testprednb_over,test$y)
```

```
##       y_pred
## y_true   N   Y
##      N 849 229
##      Y 101  92
```

```
acc_nb_over=Accuracy(testprednb_over,test$y)*100
prec_nb_over=Precision(test$y,testprednb_over,positive = "Y")*100
rec_nb_over=Recall(test$y,testprednb_over,positive = "Y")*100
spec_nb_over=Specificity(test$y,testprednb_over,positive = "Y")*100
f1_nb_over=F1_Score(test$y,testprednb_over,positive = "Y")*100
```

Let's build a model with Naive Bayes classifier using under sampling dataset.

```
nb_underfit=naiveBayes(y~.,data=under)
nb_underfit
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##   N   Y
## 0.5 0.5
##
## Conditional probabilities:
##    x1
## Y           F         M
##   N 0.6053215 0.3946785
##   Y 0.4434590 0.5565410
##
##    x2
## Y       [,1]     [,2]
##   N 48.61863 8.308818
##   Y 54.46341 7.875157
##
##    x3
## Y           1         2         3         4
##   N 0.3636364 0.3547672 0.1773836 0.1042129
##   Y 0.4878049 0.2372506 0.1552106 0.1197339
##
##    x4
## Y           N         Y
##   N 0.5188470 0.4811530
##   Y 0.4722838 0.5277162
##
##    x5
## Y        [,1]     [,2]
##   N  8.982289 12.00943
##   Y 11.044353 13.30322
##
##    x6
## Y            N          Y
##   N 0.98669623 0.01330377
##   Y 0.93791574 0.06208426
##
##    x7
## Y            N          Y
##   N 0.99556541 0.00443459
##   Y 0.98004435 0.01995565
##
##    x8
## Y           N         Y
##   N 0.7605322 0.2394678
##   Y 0.4833703 0.5166297
##
##    x9
## Y            N          Y
##   N 0.98004435 0.01995565
##   Y 0.94013304 0.05986696
##
##    x10
## Y       [,1]     [,2]
##   N 233.2367 44.27371
##   Y 244.3033 44.58329
##
##    x11
## Y       [,1]     [,2]
##   N 130.0543 20.06099
##   Y 143.4834 25.54315
##
##    x12
## Y       [,1]     [,2]
##   N 81.97561 11.51812
##   Y 86.94900 13.23483
##
##    x13
## Y       [,1]     [,2]
##   N 25.68450 3.856457
##   Y 26.34019 4.110929
##
##    x14
## Y       [,1]     [,2]
##   N 76.05322 12.07704
##   Y 76.84896 12.29833
##
##    x15
## Y       [,1]     [,2]
##   N 80.77880 13.19162
##   Y 89.78219 41.45727
```

```
summary(nb_underfit)
```

```
##           Length Class  Mode
## apriori    2     table  numeric
## tables    15     -none- list
## levels     2     -none- character
## isnumeric 15     -none- logical
## call       4     -none- call
```

Let's perform predictions and evaluate our model using confusion matrix.

```
testprednb_under=predict(nb_underfit,newdata = test)
length(testprednb_under)
```

```
## [1] 1271
```

```
#confusion matrix
ConfusionMatrix(testprednb_under,test$y)
```

```
##       y_pred
## y_true   N   Y
##      N 845 233
##      Y 101  92
```

```
acc_nb_under=Accuracy(testprednb_under,test$y)*100
prec_nb_under=Precision(test$y,testprednb_under,positive = "Y")*100
rec_nb_under=Recall(test$y,testprednb_under,positive = "Y")*100
spec_nb_under=Specificity(test$y,testprednb_under,positive = "Y")*100
f1_nb_under=F1_Score(test$y,testprednb_under,positive = "Y")*100
```

Let's build a model with Naive Bayes classifier using both over and under sampling dataset.

```
nb_bothfit=naiveBayes(y~.,data=both)
nb_bothfit
```

```
## 
## Naive Bayes Classifier for Discrete Predictors
## 
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
## 
## A-priori probabilities:
## Y
##         N         Y 
## 0.4890462 0.5109538 
## 
## Conditional probabilities:
##    x1
## Y           F         M
##   N 0.5802895 0.4197105
##   Y 0.4478892 0.5521108
## 
##    x2
## Y       [,1]     [,2]
##   N 49.15024 8.513083
##   Y 54.39710 7.872119
## 
##    x3
## Y           1         2         3         4
##   N 0.3914542 0.3108201 0.1757409 0.1219848
##   Y 0.4940633 0.2288918 0.1550132 0.1220317
## 
##    x4
## Y           N         Y
##   N 0.5024121 0.4975879
##   Y 0.4610818 0.5389182
## 
##    x5
## Y        [,1]     [,2]
##   N  9.150962 12.24480
##   Y 10.833779 12.89332
## 
##    x6
## Y            N          Y
##   N 0.98001378 0.01998622
##   Y 0.93469657 0.06530343
## 
##    x7
## Y             N           Y
##   N 0.995175741 0.004824259
##   Y 0.980211082 0.019788918
## 
##    x8
## Y           N         Y
##   N 0.7325982 0.2674018
##   Y 0.4775726 0.5224274
## 
##    x9
## Y            N          Y
##   N 0.97794624 0.02205376
##   Y 0.94129288 0.05870712
##
##    x10
## Y       [,1]     [,2]
##   N 232.6289 44.24628
##   Y 243.5822 44.52495
##
##    x11
## Y       [,1]     [,2]
##   N 130.1313 20.00298
##   Y 144.4545 25.63377
##
##    x12
## Y       [,1]     [,2]
##   N 81.68952 11.27759
##   Y 87.19294 13.13977
##
##    x13
## Y       [,1]     [,2]
##   N 25.66671 3.899304
##   Y 26.32305 4.088908
##
##    x14
## Y       [,1]     [,2]
##   N 75.40731 11.58662
##   Y 76.79000 12.51644
##
##    x15
## Y       [,1]     [,2]
##   N 81.12064 17.27532
##   Y 90.66927 42.80947
```

```
summary(nb_bothfit)
```

```
##           Length Class  Mode
## apriori    2     table  numeric
## tables    15     -none- list
## levels     2     -none- character
## isnumeric 15     -none- logical
## call       4     -none- call
```

Let's perform predictions and evaluate our model using confusion matrix.

```
testprednb_both=predict(nb_bothfit,newdata = test)
length(testprednb_both)
```

```
## [1] 1271
```

```
#confusion matrix
ConfusionMatrix(testprednb_both,test$y)
```

```
##       y_pred
## y_true   N   Y
##      N 852 226
##      Y 104  89
```

```
acc_nb_both=Accuracy(testprednb_both,test$y)*100
prec_nb_both=Precision(test$y,testprednb_both,positive = "Y")*100
rec_nb_both=Recall(test$y,testprednb_both,positive = "Y")*100
spec_nb_both=Specificity(test$y,testprednb_both,positive = "Y")*100
f1_nb_both=F1_Score(test$y,testprednb_both,positive = "Y")*100
```

Let's build a model with Naive Bayes classifier using synthetic dataset.

```
nb_syndfit=naiveBayes(y~.,data=syndata)
nb_syndfit
```

```
## 
## Naive Bayes Classifier for Discrete Predictors
## 
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
## 
## A-priori probabilities:
## Y
##      N      Y 
## 0.5025 0.4975 
## 
## Conditional probabilities:
##    x1
## Y           F         M
##   N 0.6004975 0.3995025
##   Y 0.4482412 0.5517588
## 
##    x2
## Y       [,1]     [,2]
##   N 48.97743 9.481553
##   Y 54.21946 9.076010
## 
##    x3
## Y           1         2         3         4
##   N 0.3810945 0.3447761 0.1716418 0.1024876
##   Y 0.4929648 0.2351759 0.1442211 0.1276382
## 
##    x4
## Y           N         Y
##   N 0.5154229 0.4845771
##   Y 0.4904523 0.5095477
## 
##    x5
## Y        [,1]     [,2]
##   N  8.665109 13.32594
##   Y 10.742061 15.28850
## 
##    x6
## Y            N          Y
##   N 0.97910448 0.02089552
##   Y 0.93115578 0.06884422
## 
##    x7
## Y             N           Y
##   N 0.996517413 0.003482587
##   Y 0.979899497 0.020100503
## 
##    x8
## Y           N         Y
##   N 0.7333333 0.2666667
##   Y 0.4723618 0.5276382
## 
##    x9
## Y            N          Y
##   N 0.98855721 0.01144279
##   Y 0.93618090 0.06381910
##
##    x10
## Y       [,1]     [,2]
##   N 234.7126 47.56355
##   Y 244.7614 50.88094
##
##    x11
## Y       [,1]     [,2]
##   N 130.3203 22.50790
##   Y 143.3178 29.07172
##
##    x12
## Y       [,1]     [,2]
##   N 81.89498 12.76779
##   Y 87.09272 15.11773
##
##    x13
## Y       [,1]     [,2]
##   N 25.67047 4.466229
##   Y 26.36651 4.800696
##
##    x14
## Y       [,1]     [,2]
##   N 75.52216 13.47080
##   Y 76.52907 14.12276
##
##    x15
## Y       [,1]     [,2]
##   N 80.49815 16.76829
##   Y 89.63396 47.79267
```

```
summary(nb_syndfit)
```

```
##           Length Class  Mode
## apriori    2     table  numeric
## tables    15     -none- list
## levels     2     -none- character
## isnumeric 15     -none- logical
## call       4     -none- call
```

Let's perform predictions and evaluate our model using confusion matrix.

```
testprednb_syn=predict(nb_syndfit,newdata = test)
length(testprednb_syn)
```

```
## [1] 1271
```

```
#confusion matrix
ConfusionMatrix(testprednb_syn,test$y)
```

```
##       y_pred
## y_true   N   Y
##      N 888 190
##      Y 110  83
```

```
acc_nb_syn=Accuracy(testprednb_syn,test$y)*100
prec_nb_syn=Precision(test$y,testprednb_syn,positive = "Y")*100
rec_nb_syn=Recall(test$y,testprednb_syn,positive = "Y")*100
spec_nb_syn=Specificity(test$y,testprednb_syn,positive = "Y")*100
f1_nb_syn=F1_Score(test$y,testprednb_syn,positive = "Y")*100
```

Now that we have all the necessary information regarding the predictions of different classification techniques, all we need to do is to compare the different techniques based on different solutions for solving data imbalance.

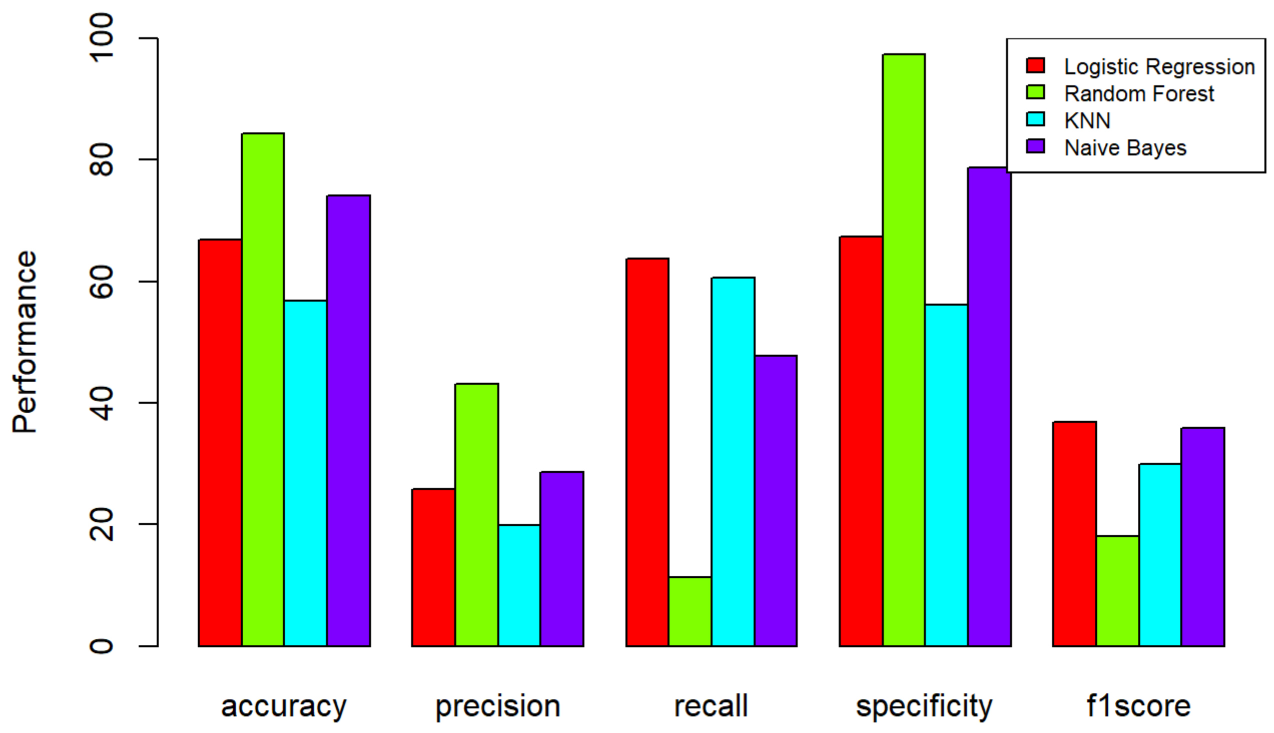
```
#oversampling performance evaluations (in %)
accuracy=c(acc_lr_over,acc_rf_over,acc_knn_over,acc_nb_over)
precision=c(prec_lr_over,prec_rf_over,prec_knn_over,prec_nb_over)
recall=c(rec_lr_over,rec_rf_over,rec_knn_over,rec_nb_over)
specificity=c(spec_lr_over,spec_rf_over,spec_knn_over,spec_nb_over)
f1score=c(f1_lr_over,f1_rf_over,f1_knn_over,f1_nb_over)

oversampling_evaluations=data.frame(accuracy,precision,recall,specificity,f1score)
rownames(oversampling_evaluations)=c("Logistic Regression","Random Forest","KNN","Naive Baye
s")
oversampling_evaluations
```

```
##                     accuracy precision   recall specificity  f1score
## Logistic Regression 66.79780  25.89474 63.73057    67.34694 36.82635
## Random Forest       84.26436  43.13725 11.39896    97.30983 18.03279
## KNN                 56.88434  19.86418 60.62176    56.21521 29.92327
## Naive Bayes         74.03619  28.66044 47.66839    78.75696 35.79767
```

```
performance1=as.matrix(oversampling_evaluations)

barplot(performance1,beside=TRUE,col=rainbow(4),ylim=c(0,100),ylab = "Performance")
legend("topright",legend = c("Logistic Regression","Random Forest","KNN","Naive Bayes"),cex =
0.7,fill = rainbow(4))
```



The precision of each and every technique came down quite a bit thus affecting the F1 Score as well. But based on the other performance measures, Logistic Regression had a very decent overall scores and we can say, that it's the based among all the classification techniques.

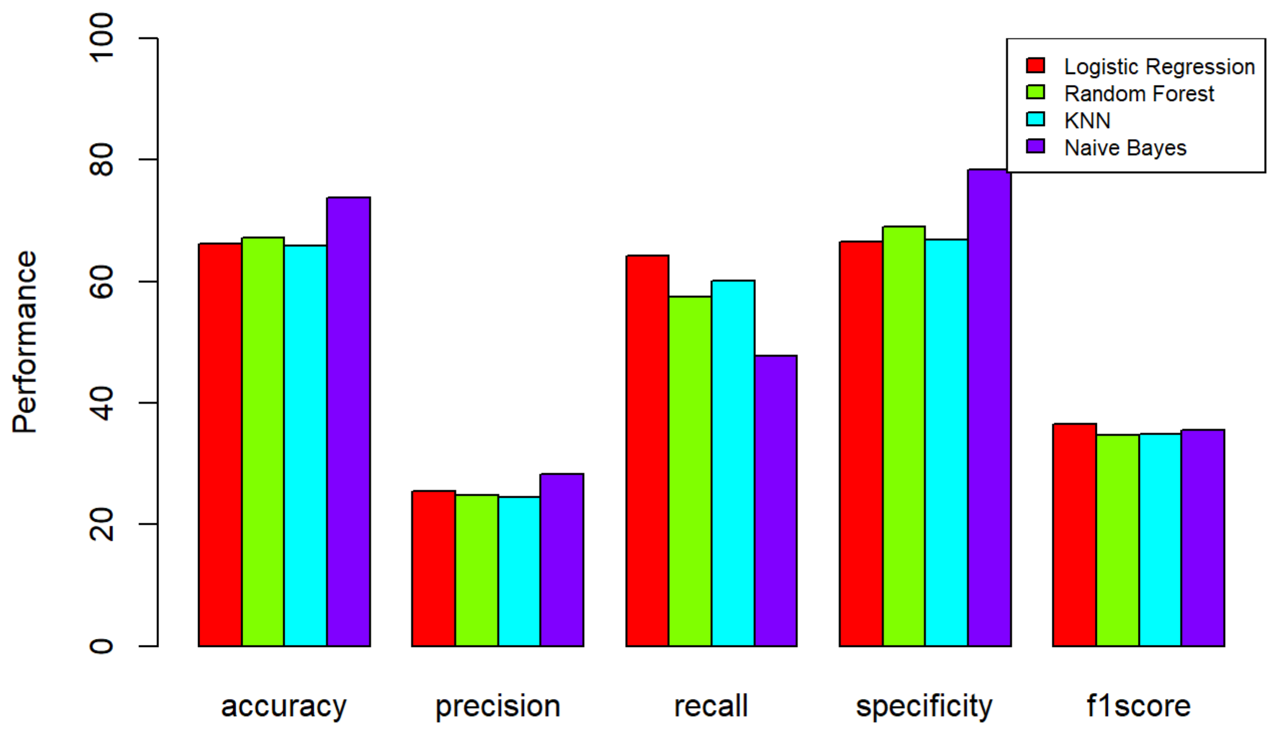
```
#undersampling performance evaluations
accuracy=c(acc_lr_under,acc_rf_under,acc_knn_under,acc_nb_under)
precision=c(prec_lr_under,prec_rf_under,prec_knn_under,prec_nb_under)
recall=c(rec_lr_under,rec_rf_under,rec_knn_under,rec_nb_under)
specificity=c(spec_lr_under,spec_rf_under,spec_knn_under,spec_nb_under)
f1score=c(f1_lr_under,f1_rf_under,f1_knn_under,f1_nb_under)

undersampling_evaluations=data.frame(accuracy,precision,recall,specificity,f1score)
rownames(undersampling_evaluations)=c("Logistic Regression","Random Forest","KNN","Naive Baye
s")
undersampling_evaluations
```

```
##                     accuracy precision   recall specificity  f1score
## Logistic Regression 66.16837  25.56701 64.24870    66.51206 36.57817
## Random Forest       67.19119  24.88789 57.51295    68.92393 34.74178
## KNN                 65.85366  24.52431 60.10363    66.88312 34.83483
## Naive Bayes         73.72148  28.30769 47.66839    78.38590 35.52124
```

```
performance2=as.matrix(undersampling_evaluations)

barplot(performance2,beside=TRUE,col=rainbow(4),ylim=c(0,100),ylab = "Performance")
legend("topright",legend = c("Logistic Regression","Random Forest","KNN","Naive Bayes"),cex =
0.7,fill = rainbow(4))
```



Once again, the value of precision is very low, thus affecting the value of F1 Score as well. But based on the other performance measures, Logistic Regression had a very decent overall scores and we can say, that it's the based among all the classification techniques.

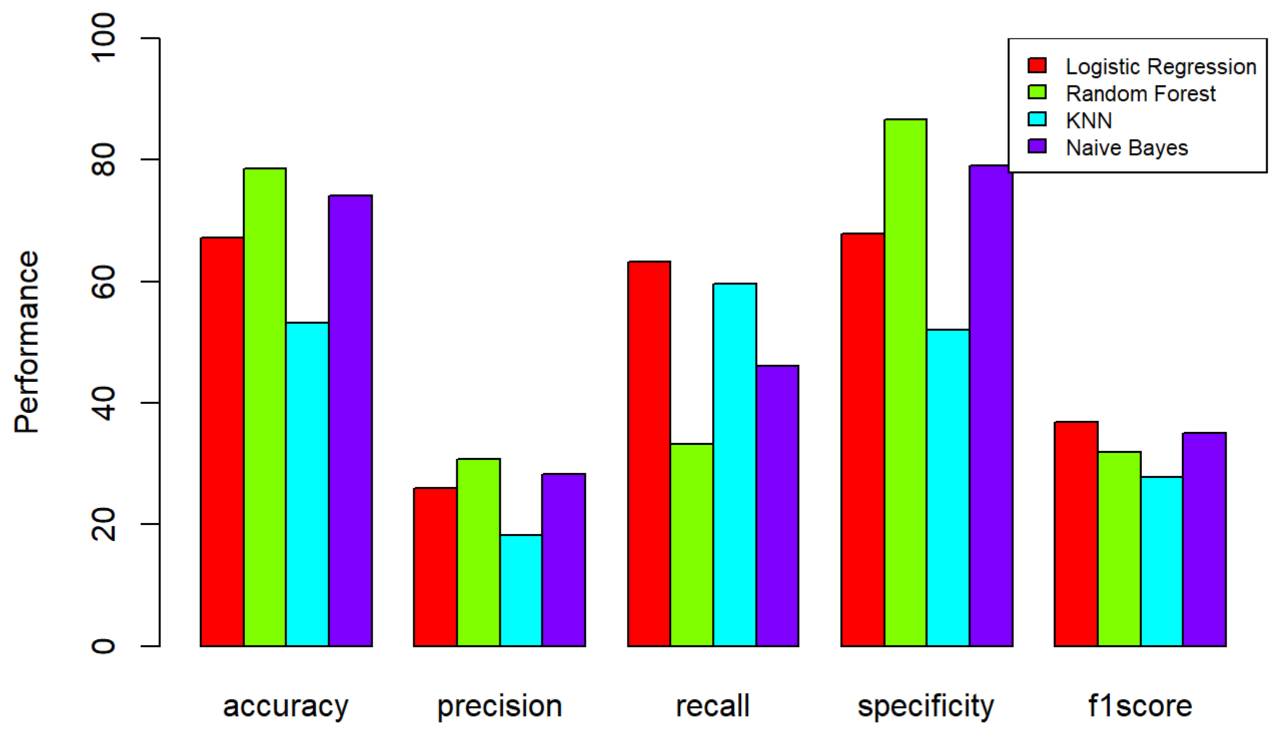
```
#both over and undersampling performance evaluation
accuracy=c(acc_lr_both,acc_rf_both,acc_knn_both,acc_nb_both)
precision=c(prec_lr_both,prec_rf_both,prec_knn_both,prec_nb_both)
recall=c(rec_lr_both,rec_rf_both,rec_knn_both,rec_nb_both)
specificity=c(spec_lr_both,spec_rf_both,spec_knn_both,spec_nb_both)
f1score=c(f1_lr_both,f1_rf_both,f1_knn_both,f1_nb_both)

bothsampling_evaluations=data.frame(accuracy,precision,recall,specificity,f1score)
rownames(bothsampling_evaluations)=c("Logistic Regression","Random Forest","KNN","Naive Baye
s")
bothsampling_evaluations
```

```
##                     accuracy precision   recall specificity  f1score
## Logistic Regression 67.11251  26.01279 63.21244    67.81076 36.85801
## Random Forest       78.52085  30.76923 33.16062    86.64193 31.92020
## KNN                 53.18647  18.19620 59.58549    52.04082 27.87879
## Naive Bayes         74.03619  28.25397 46.11399    79.03525 35.03937
```

```
performance3=as.matrix(bothsampling_evaluations)

barplot(performance3,beside=TRUE,col=rainbow(4),ylim=c(0,100),ylab = "Performance")
legend("topright",legend = c("Logistic Regression","Random Forest","KNN","Naive Bayes"),cex =
0.7,fill = rainbow(4))
```



Once again, the value of precision is very low, thus affecting the value of F1 Score as well. But based on the other performance measures, Logistic Regression had a very decent overall scores and we can say, that it's the based among all the classification techniques.

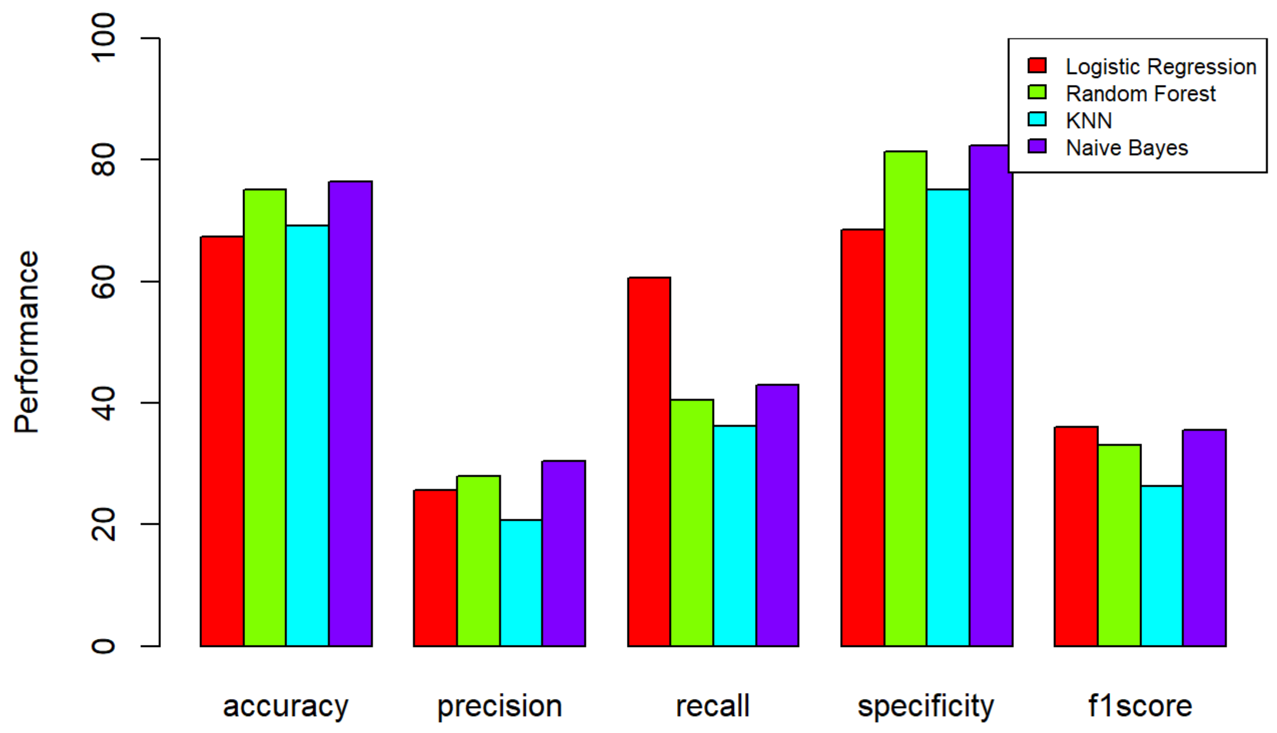
```
#synthetic data performance evaluation
accuracy=c(acc_lr_syn,acc_rf_syn,acc_knn_syn,acc_nb_syn)
precision=c(prec_lr_syn,prec_rf_syn,prec_knn_syn,prec_nb_syn)
recall=c(rec_lr_syn,rec_rf_syn,rec_knn_syn,rec_nb_syn)
specificity=c(spec_lr_syn,spec_rf_syn,spec_knn_syn,spec_nb_syn)
f1score=c(f1_lr_syn,f1_rf_syn,f1_knn_syn,f1_nb_syn)

syndata_evaluations=data.frame(accuracy,precision,recall,specificity,f1score)
rownames(syndata_evaluations)=c("Logistic Regression","Random Forest","KNN","Naive Bayes")
syndata_evaluations
```

```
##                     accuracy precision   recall specificity  f1score
## Logistic Regression 67.34854  25.65789 60.62176    68.55288 36.05547
## Random Forest       75.13769  27.95699 40.41451    81.35436 33.05085
## KNN                 69.15814  20.64897 36.26943    75.04638 26.31579
## Naive Bayes         76.39654  30.40293 43.00518    82.37477 35.62232
```

```
performance4=as.matrix(syndata_evaluations)

barplot(performance4,beside=TRUE,col=rainbow(4),ylim=c(0,100),ylab = "Performance")
legend("topright",legend = c("Logistic Regression","Random Forest","KNN","Naive Bayes"),cex =
0.7,fill = rainbow(4))
```



Once again, the value of precision is very low, thus affecting the value of F1 Score as well. But based on the other performance measures, Logistic Regression had a very decent overall scores and we can say, that it's the based among all the classification techniques.