PREDICTING THE RISK OF HEART DISEASE: A CLASSIFICATION PROBLEM USING LOGISTIC REGRESSION, RANDOM FOREST, SUPPORT VECTOR MACHINE AND NAÏVE BAYES CLASSIFIER

A PROJECT REPORT

Submitted by

AGNIVA KONAR

UPASYA BOSE

TIYASHA SAMANTA

Supervised by

MR. BRATIN DAS

In partial fulfillment for the award of the degree of

MASTER OF SCIENCE

IN

APPLIED STATISTICS AND ANALYTICS

IN YEAR 2021-2022

MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

NH-12 (Old NH-34) Simhat

Haringhata, Nadia 741249, West Bengal

BONAFIDE CERTIFICATE

Certified that this project report **"PREDICTING THE RISK OF HEART DISEASE: A CLASSIFICATION PROBLEM USING LOGISTIC REGRESSION, RANDOM FOREST, SUPPORT VECTOR MACHINE AND NAÏVE BAYES CLASSIFIER"** is the bonafide work of AGNIVA KONAR, UPASYA BOSE & TIYASHA SAMANTA who carried out the projectwork under my supervision.

_____

SUPERVISOR

_____

HOD, Department of Applied Science

SONASS

_____

External Examiner

ACKNOWLEDGEMENT

It is a great pleasure for us to undertake this project. Weare utterly grateful to our project guide Mr. Bratin Das.

This project would not have been completed without his

enormous help and worthy experience. Whenever we were in a fix, he was there to guide us and provide us with a perfect solution. Although, this project report hasbeen prepared with utmost care and deep routed interest, even then, we accept our flaws and

imperfection.

Date:

TABLE OF CONTENTS

SL. NO.	TITLE	PAGE NO.
1.	SUMMARY/ABSTRACT	5
2.	KEYWORDS	5
3.	INTRODUCTION	6
4.	AIM	7
5.	OBJECTIVES	7
6.	SCOPE AND NEED OF THE PROJECT	8
7.	LOGISTIC REGRESSION	8
8.	HYPOTHESIS TESTING	9
9.	TEST FOR MULTICOLLINEARITY	9
10.	RANDOM FOREST	11
11.	SUPPORT VECTOR MACHINE (SVM)	12
12.	NAÏVE BAYES CLASSIFIER	14
13.	RESEARCH METHODOLOGY	15
14.	SPLITTING OF DATASET	15
15.	MEASURES OF SOLVING DATA IMBALANCE	16
16.	OVER SAMPLING	16
17.	UNDER SAMPLING	18
18.	BOTH OVER AND UNDER SAMPLING	19
19.	SYNTHETIC DATA GENERATION	20
20.	PROCEDURE	21
21.	OUTCOMES	42
22.	RESOURCES	46
23.	IMPORTANT LINKS	46

SUMMARY/ABSTRACT

The given data consisted of some basic information about a person and numerous other factors that can be a reason for heart disease of a patient. These factors include smoking, alcohol consumption, etc. It can be observed that the given data at hand is a multivariate data. Our job here is to build a model using different classification techniques that will help us to classify whether a person has the risk of having a heart disease within the upcoming ten years based on the given data at hand. We are going to use classification techniques such as Logistic Regression, Random Forest and Support Vector Machine (SVM) to do so. We need to solve for data imbalance, if any, and then we will move on to building our classification model. We will use the model for predictions and we will evaluate our model performance using some performance measure. Basically, we will compare which technique is giving us efficient predictions with our dataset.

KEYWORDS

1. Multivariate Data
2. Data Imbalance
3. Over Sampling
4. Under Sampling
5. Both Over and Under Sampling
6. Synthetic Data Generation
7. Logistic Regression
8. Random Forest
9. K Nearest Neighbors (KNN)
10. Confusion Matrix
11. Accuracy
12. Sensitivity or Recall
13. Specificity
14. Precision
15. F Measure
16. Misclassification

INTRODUCTION

The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data. In Classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups. Such as, **Yes or No, 0 or 1, Spam or Not Spam, cat or dog,** etc. Classes can be called as targets/labels or categories.

Unlike regression, the output variable of Classification is a category, not a value, such as "Green or Blue", "fruit or animal", etc.

Since the Classification algorithm is a Supervised learning technique, hence it takes labelled input data, which means it contains input with the corresponding output.

In classification algorithm, a discrete output function(y) is mapped to input variable(x). The best example of an ML classification algorithm is **Email Spam Detector**.

The main goal of the Classification algorithm is to identify the category of a given dataset, and these algorithms are mainly used to predict the output for the categorical data. The algorithm which implements the classification on a dataset is known as a classifier.

There are two types of Classifications:

- **Binary Classifier:** If the classification problem has only two possible outcomes, then it is called as Binary Classifier. **Examples:** YES or NO, SPAM or NOT SPAM, etc.
- **Multi-class Classifier:** If a classification problem has more than two outcomes, then it is called as Multi-class Classifier. **Example:** Classifications of types of flowers, Classification of types of music, etc.

In this case, we need to predict whether a person is going to have a heart disease in the coming 10 years or not, hence, this is a case of binary classification.

AIM

The aim is to classify the persons whether they have the risk of having a heart disease in the next ten years or not. These are the two classes, named as Y and N. This classification will be done based on certain factors such as smoking habits, alcohol consumption, blood pressure, etc. We are going to use four different classification techniques, namely, Logistic regression, Random Forest, K Nearest Neighbours (KNN) and Naïve Bayes Classifier to do the following. At the end, we are going to compare the four classification techniques by evaluating the performance measures of the models by creating confusion matrices for each of the technique.

OBJECTIVES

a) The primary data at hand contains non numeric columns which must beconverted to categorical data for ease of calculation.

b) The primary data may contain missing numeric values. Those values must be replaced.

c) We need to check for data imbalance and if found, we need to solve that issue, or else our classification will be biased for a particular class.

d) Test for Multicollinearity needs to be done for the case of Logistic Regression to find out which covariates have high correlation among themselves and then those covariates need to be dropped or some other technique must be opted to solve the issue of multicollinearity to ensure a better classification model.

e) For other classification techniques, no such tests need to be carried out, because the other techniques are assumption free.

f) After building out necessary classification models, we will perform predictions and evaluate the model performance based on certain measures obtained from the confusion matrix for each of the classification techniques.

g) Based on the performance evaluation, we will find out which classification technique proved to be the most fruitful for our dataset.

SCOPE AND NEED OF THE PROJECT

This project can be a very useful in the field of medical sciences. Developing an efficient model will help us properly classifying persons based on their habits and health records, whether they have the risk of heart disease or not. We can collect data for our covariates and put the values in the model and we will get the classification beforehand and the person can get treatments beforehand. We can also add other covariates such as hypertension and food habits and can observe how these covariates help a person in classifying.

LOGISTIC REGRESSION

Logistic regression is a classification algorithm used to find the probability of event success and event failure. Logistic regression is used when the dependent variable is binary (0/1, True/False, Yes/No) in nature. Logit function is used as a link function in a binomial distribution. Logistic regression is also known as **Binomial logistics regression**. It is based on sigmoid function where output is probability and input can be from -infinity to +infinity. Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labelled "0" and "1". In the logistic model, the log odds (the logarithm of the odds) for the value labelled "1" is a linear combination of one or more independent variables ("predictors"); the independent variables can each be a binary variable (two classes, coded by an indicator variable) or a continuous variable (any real value). The corresponding probability of the value labelled "1" can vary between 0 (certainly the value "0") and 1 (certainly the value "1"), hence the labelling; the function that converts log-odds to probability is the logistic function, hence the name. The unit of measurement for the log-odds scale is called a logit, from ***log****istic un****it***, hence the alternative names. Analogous models with a different sigmoid function instead of the logistic function can also be used, such as the probit model; the defining characteristic of the logistic model is that increasing one of the independent variables multiplicatively scales the odds of the given

outcome at a *constant* rate, with each independent variable having its own parameter; for a binary dependent variable this generalizes the odds ratio.

In RStudio, logistic regression is carried out using the glm function as it is a form of generalised linear model, and hence, the abbreviation.

Since, logistic regression is a form of regression analysis, hence, it is important to check whether the covariates or the predictor variables are independent of each other and if not, certain measures need to be taken to establish independency among the covariates.

```
#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 = "binomi
al")
summary(model2)$deviance
```

```
## [1] 1045.396
```

HYPOTHESIS TESTING

Test for Multicollinearity

Multicollinearity is a situation which occurs when the predictor variables in a multiple linear regression model are highly correlated to each other. Inother words, it basically refers to an incident where an independent variable of a regression model can be linearly predicted using other covariates. Presence of multicollinearity in a data isn't a good sign as it will affect the statistical inference made from the data. Those inference won't be reliable enough. It also lowers the accuracy of the estimate of coefficients of our model, and hence it becomes difficult to find out the significant factors from their corresponding p values. However, these problems arise when the multicollinearity is very high. If the data shows moderate multicollinearity, then there's no need to remove/ reduce it.

Way of testing Multicollinearity

One very popular way of testing for multicollinearity is by finding the Variance Inflation Factor (VIF). Using this, we can find out which

predictor variables are affected by multicollinearity and the strength of correlation and hence we can act accordingly. VIF has a lower limit of 1 but has no upper limit. For calculating VIF, we need to plot a regression model using a predictor variable as the response and the rest of the predictor variables as the covariates. Then we need to find the

$$VIF_i = \frac{1}{1 - R_i^2} = \frac{1}{Tolerance}$$

coefficient of determination of the following regression model. Then VIF is calculated as:

If the VIF has a value of 1, then it means that there is no multicollinearity,or in simpler terms, there is no correlation present between that independent variable and any others. Hence, the corresponding predictor variable doesn't need to be dropped.

If the VIF value lies between 1 and 10, then it suggests the presence of moderate multicollinearity. Hence, even in this case, the correspondingpredictor variable doesn't need to be dropped.

But, if the VIF value comes out as greater than 10, then that indicates highmulticollinearity and that will lead to erroneous results and hence that predictor variable needs to be dropped.

We carry out the test of multicollinearity using the vif() in RStudio. The result is given as:

```
##         GVIF Df GVIF^(1/(2*Df))
## x1  1.256219  1        1.120812
## x2  1.373772  1        1.172080
## x3  1.140994  3        1.022227
## x4  2.733914  1        1.653455
## x5  2.852463  1        1.688924
## x6  1.086776  1        1.042486
## x7  1.014154  1        1.007052
## x8  2.059961  1        1.435256
## x9  1.602490  1        1.265895
## x10 1.059578  1        1.029358
## x11 3.425263  1        1.850747
## x12 2.771149  1        1.664677
## x13 1.175202  1        1.084067
## x14 1.091909  1        1.044944
## x15 1.629200  1        1.276401
```

Conclusion of the test:

We can observe, that the vif values for each covariate is lesser than 10.Hence, we can conclude that there is only presence of moderate multicollinearity in our dataset and thus, we don't have to drop any covariates.

RANDOM FOREST

A random forest is a machine learning technique that's used to solve regression and classification problems. It utilizes ensemble learning, which is a technique that combines many classifiers to provide solutions to complex problems.

A random forest algorithm consists of many decision trees. The 'forest' generated by the random forest algorithm is trained through bagging or bootstrap aggregating. Bagging is an ensemble meta-algorithm that improves the accuracy of machine learning algorithms. The algorithm establishes the outcome based on the predictions of the decision trees. It predicts by taking the average or mean of the output from various trees. Increasing the number of trees increases the precision of the outcome. It eradicates the limitations of a decision tree algorithm. It reduces the overfitting of datasets and increases precision.

Classification in random forests employs an ensemble methodology to attain the outcome. The training data is fed to train various decision trees. This dataset consists of observations and features that will be selected randomly during the splitting of nodes.

A random forest system relies on various decision trees. Every decision tree consists of decision nodes, leaf nodes, and a root node. The leaf node of each tree is the final output produced by that specific decision tree. The selection of the final output follows the majority-voting system. In this case,

the output chosen by the majority of the decision trees becomes the final output of the random forest system.

In RStudio, Random Forest is carried out by using the randomForest function.

```
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
```

K NEAREST NEIGHBOURS (KNN)

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. KNN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

KNN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using this algorithm. It can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data. It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset. KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

```r
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, ...
```

NAÏVE BAYES CLASSIFIER

Naive Bayes classifiers are a collection of classification algorithms based on **Bayes' Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e., every pair of features being classified is independent of each other. This is the reason why the Naive Bayes algorithm is called "Naive",

Naive Bayes is a Supervised Non-linear classification algorithm. Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (Naive) independence assumptions between the features or variables

Naïve Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum likelihood training can be done by evaluating a closed-form expression which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

The fundamental Naive Bayes assumption is that each feature makes an:

- independent
- equal

contribution to the outcome.

Despite their naive design and apparently oversimplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations. An advantage of naive Bayes is that it only requires a small number of training data to estimate the parameters necessary for classification.

```
model19<- naiveBayes(y~.,data = over)
model19
```

```
##
## 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
```

RESEARCH METHODOLOGY

We have obtained the dataset from Kaggle. The data was in the form of Excel spreadsheet containing 4238 rows and 16 columns. Before starting to build our classification model, we need to do some basic data wrangling.

The dataset contained several missing values and NA values. Those values were replaced by the mean of the particular columns. This step is known as data conversion. It was a pretty well constructed data and hence, the step of data conversion wasn't time consuming at all.

SPLITTING OF DATASET:

Before moving on to the data cleansing, we first have to split the dataset into training and testing dataset. This splitting must be done randomly.

We split the dataset in the ratio of 70:30. The 70% of the dataset is named as the training dataset whereas the remaining 30% is named as the testing dataset.

All kinds of data cleansing must be done only on the training dataset, because this is the data that we are going to feed the model for the sake of it's learning. Once the model is ready, then we will perform the

predictionon the basis of testing dataset, to look how well it is performing when it faces an unknown dataset.

```
#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
```

As we can observe, that after splitting our dataset in the ratio of 70:30, there is a huge data imbalance in our training dataset. The N class is clearly dominant over the Y class and this imbalance will surely cause biased towards the N class.

Hence, we need to opt for some measures on the training set to solve this issue of data imbalance.

MEASURES OF SOLVING DATA IMBALANCE

A) OVER SAMPLING

Random oversampling involves randomly duplicating examples from the minority class and adding them to the training dataset.

Examples from the training dataset are selected randomly with replacement. This means that examples from the minority class can be chosen and added to the new "*more balanced*" training dataset multiple times; they are selected from the original training dataset, added to the new training dataset, and then returned or "*replaced*" in the original dataset, allowing them to be selected again.

This technique can be effective for those machine learning algorithms that are affected by a skewed distribution and where multiple duplicate examples for a given class can influence the fit of the model. This might include algorithms that iteratively learn coefficients, like artificial neural networks that use stochastic gradient descent. It can also affect models that seek good splits of the data, such as support vector machines and decision trees.

It might be useful to tune the target class distribution. In some cases, seeking a balanced distribution for a severely imbalanced dataset can cause affected algorithms to overfit the minority class, leading to increased generalization error. The effect can be better performance on the training dataset, but worse performance on the holdout or test dataset.

Random Oversampling is done in the following manner in RStudio:

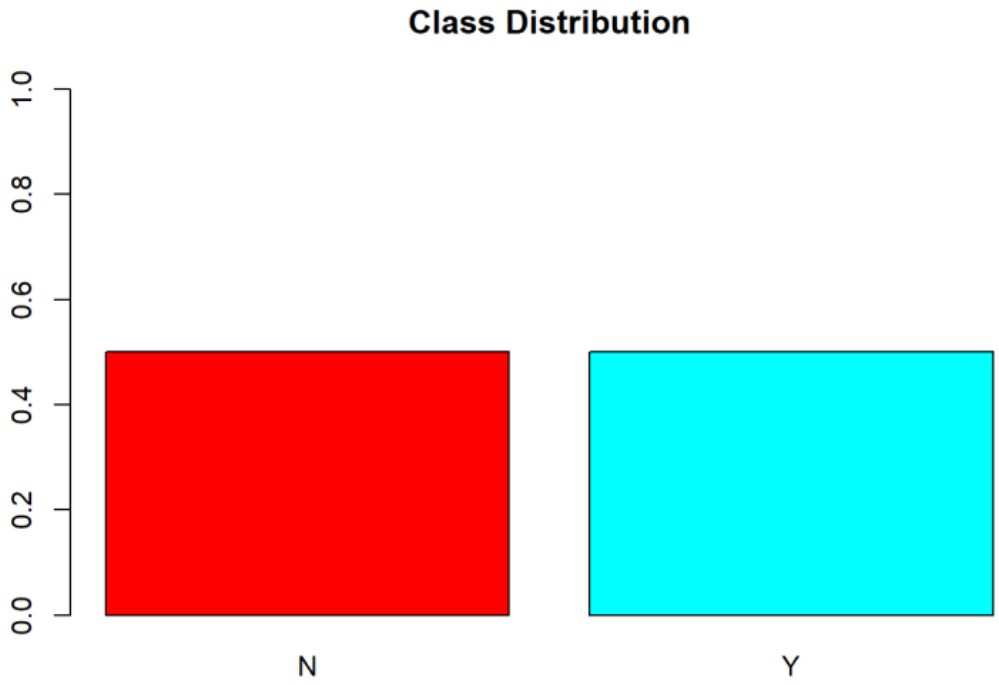
```
#oversampling
over<-ovun.sample(y~.,data = train, method = "over",N=5032)$data
dim(over)
```

```
## [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 after performing random oversampling, the sample size has increased significantly and does the count of Y class values. After performing oversampling, Y has become the dominant class but this won't cause data imbalance because our machine will be able to learn a lot about the N class as well and hence, it will not be biased towards a particular class at the time of classification.

B) UNDER SAMPLING

Random under sampling involves randomly selecting examples from the majority class to delete from the training dataset.

This has the effect of reducing the number of examples in the majority class in the transformed version of the training dataset. This process can be repeated until the desired class distribution is achieved, such as an equal number of examples for each class.

This approach may be more suitable for those datasets where there is a class imbalance although a sufficient number of examples in the minority class, such a useful model can be fit.

A limitation of under sampling is that examples from the majority class are deleted that may be useful, important, or perhaps critical to fitting a robust decision boundary. Given that examples are deleted randomly, there is no way to detect or preserve "*good*" or more information-rich examples from the majority class.

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

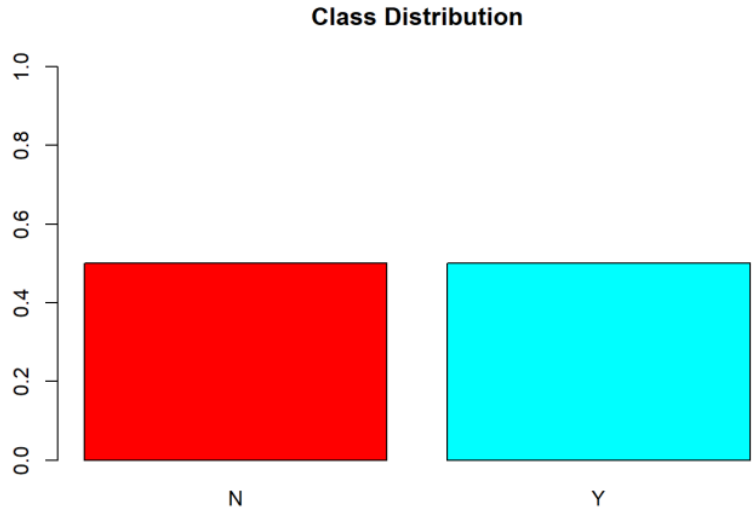
```
## [1] 902  16
```

```
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 after performing under sampling, the proportion of both the classes becomes equal and thus, the machine will learn to distinguish for the classes properly and learn about them. Let's visualise the class proportion with a bar plot.



There’s no chance for the machine to be biased towards a particular class at the time of classification.

C) BOTH OVER AND UNDER SAMPLING

Combining both random sampling methods can occasionally result in overall improved performance in comparison to the methods being performed in isolation. The concept is that we can apply a modest amount of oversampling to the minority class, which improves the bias to the minority class examples, whilst we also perform a modest amount of under sampling on the majority class to reduce the bias on the majority class examples.

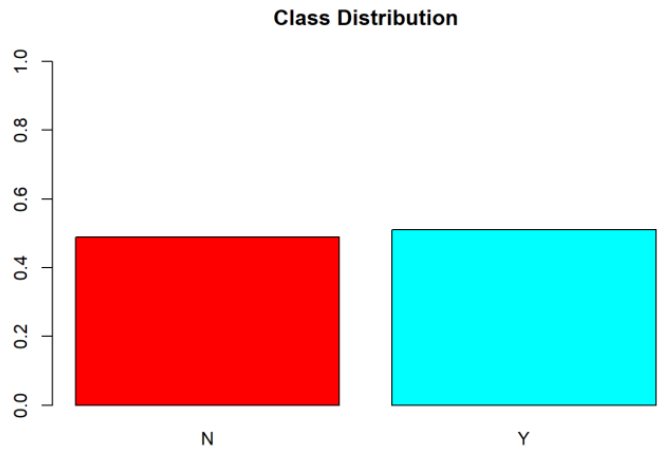
```
#both over and under sampling
both<-ovun.sample(y~.,data = train,method = "both",p=0.5,seed = 222,N=2967)$data
dim(both)
```

```
## [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")
```



We can observe that the number of Y values are slightly greater than the N values but the thing to be noticed is, that the total number of observations are same as that of the training dataset unlike the cases of over sampling and under sampling.

D) SYNTHETIC DATA GENERATION

This technique synthesises new minority instances between existing (real) minority instances. It creates new synthetic instances according to the neighbourhood of each example of the minority class. It is a widespread solution used in imbalanced classification problems and it is available in several commercial and open source software packages. It does not replicate data in the general region of the minority instances, but on the exact locations. As such it can cause models to overfit to the data. In this technique, synthetic data is used rather than replacement.

In RStudio, this is done by using the ROSE function.

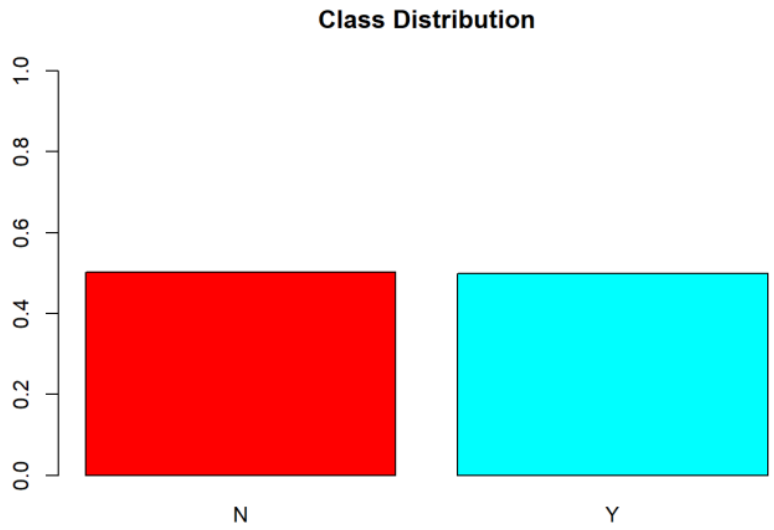
```
syndata<-ROSE(y~.,data = train,N=4238,seed = 111)$data
dim(syndata)
```

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

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

```
##
##         N         Y
## 0.5018877 0.4981123
```

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



The problem of data imbalance is solved by the technique of synthetic data generation and hence, the code will not be biased towards a particular class during classification.

Now that we are able to create new datasets that don't have the issue of data imbalance, we can use these datasets to train our classification models based on different techniques and we will perform predictions based on the previously created testing dataset. We will compare the predictions and draw conclusions about the performance of our model based on certain measures.

PROCEDURE

Once we have completed all the steps mentioned in the research methodology section, we have our new datasets which will be used for primary classification model building.

The classification techniques which are mentioned above are going to be executed for each of the datasets and later their performance will be evaluated based on some certain measures.

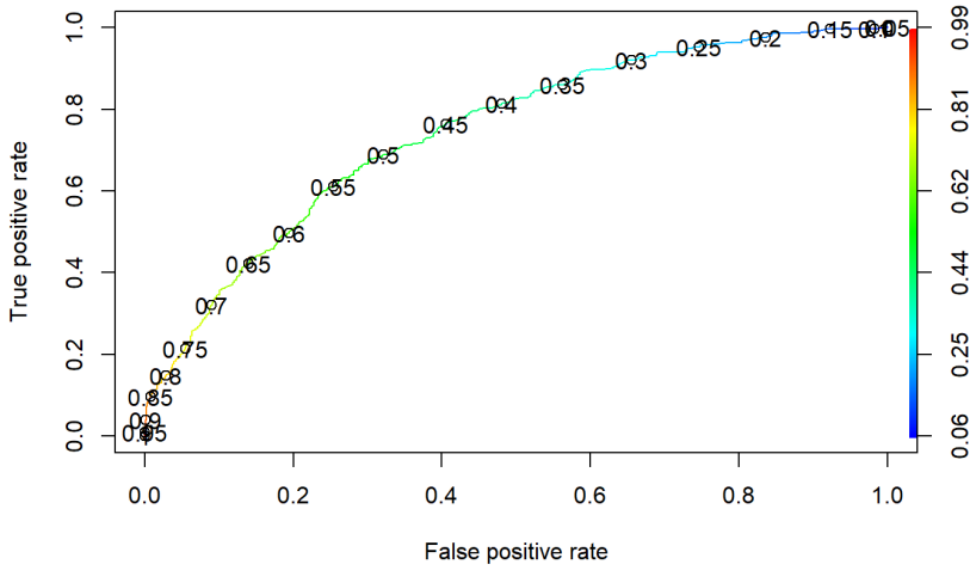
Logistic Regression with Oversampling dataset:

Step 1: Building the Logistic Regression model using glm()

```
#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
```

Step 2: Choosing the ideal cutoff value with the help of ROC Curve



Step 3: Performing predictions and generating the confusion matrix

```
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
```

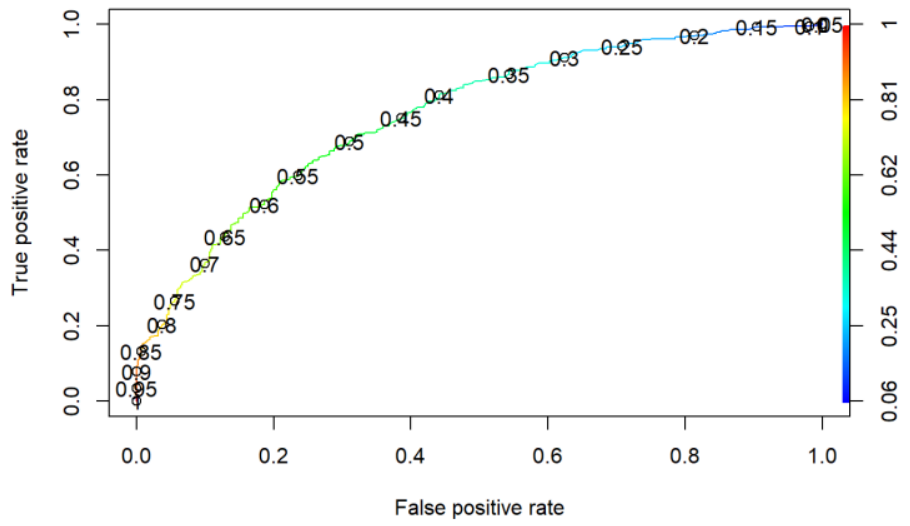
Logistic Regression with Under sampling dataset:

Step 1: Building the Logistic Regression model using glm()

```
#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 = binomial(link = "logit"))

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

Step 2: Choosing the ideal cutoff value with the help of ROC Curve




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

Step 3: Performing predictions and generating the confusion matrix

```
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
```

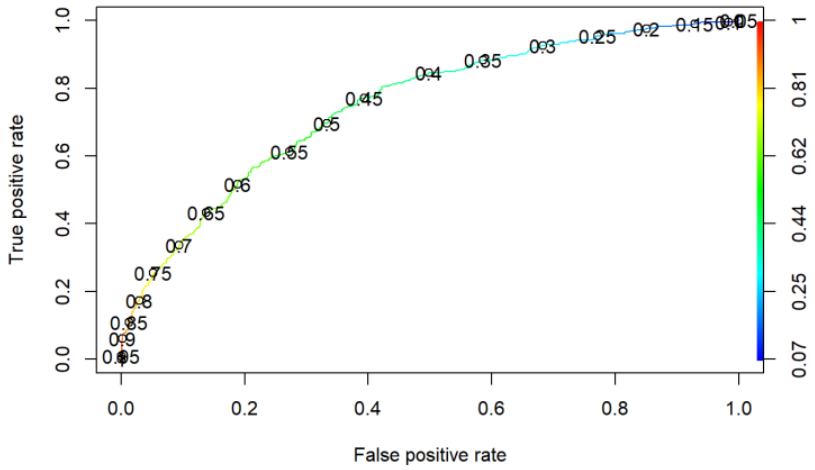
Logistic Regression with Both Over and Under sampling dataset:

Step 1: Building the Logistic Regression model using glm()

```
#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
```

Step 2: Choosing the ideal cutoff value with the help of ROC Curve




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

Step 3: Performing predictions and generating the confusion matrix

```
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
```

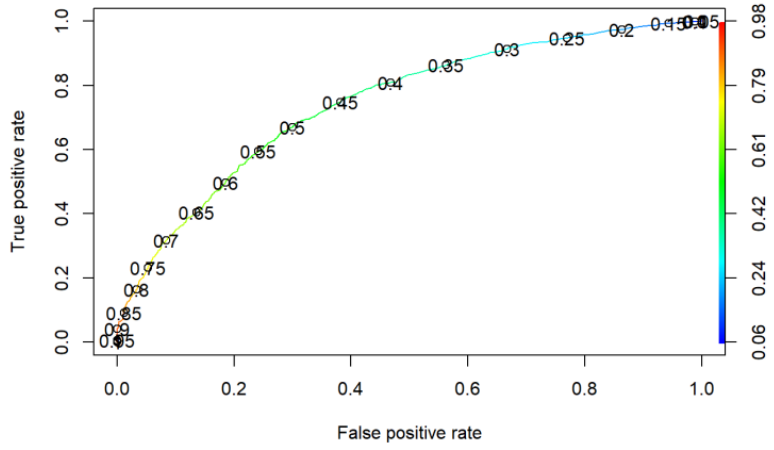
Logistic Regression with Synthetic dataset:

Step 1: Building the Logistic Regression model using glm()

```
#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 = binomial(link = "logit"))

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

Step 2: Choosing optimum cutoff value with the help of ROC Curve




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

Step 3: Performing predictions and generating the confusion matrix

```
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
```

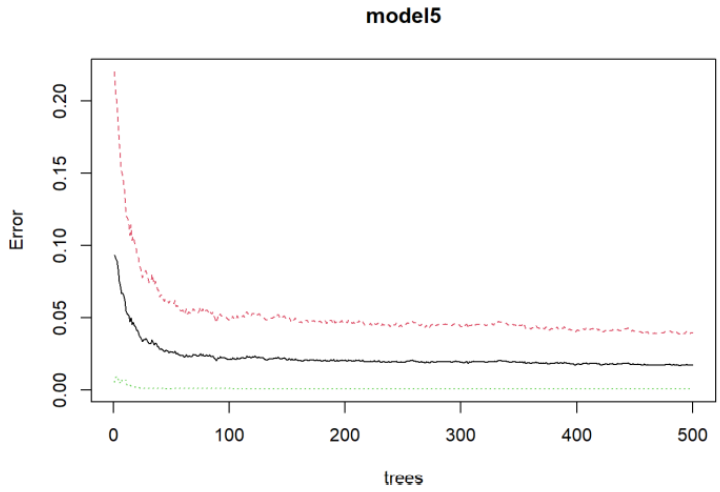
Random Forest with Oversampling dataset:

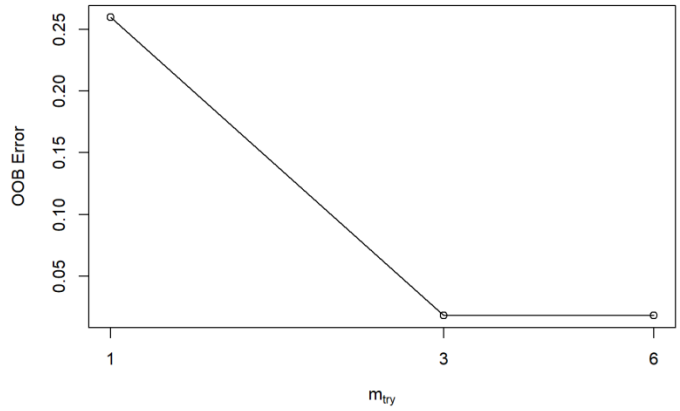
Step 1: Build a primary Random Forest model

```
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
```

Step 2: Find out the optimum value for ntree and mtry by plotting the following graphs and tune the model accordingly.






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

From the 1st graph, we can say that value of ntree = 500 is appropriate as the OOB error estimate gets saturated a lot. From the 2nd graph, we can observe that the OOB error estimate is lowest from mtry=4 and hence, this is the appropriate value of mtry. Thus, we will tune the model accordingly.

Step 3: Performing predictions and generating 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
```

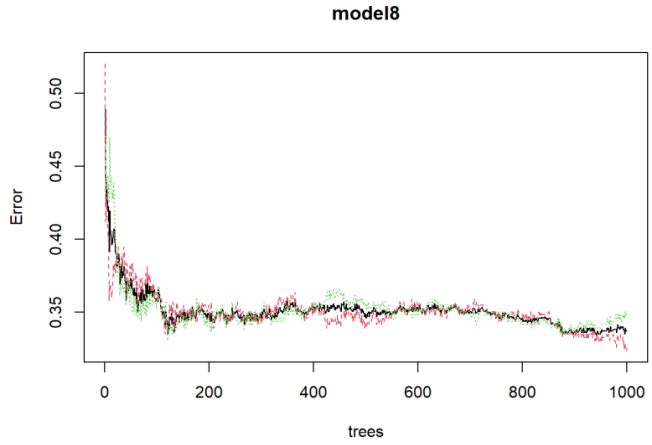
Random Forest with Under sampling dataset:

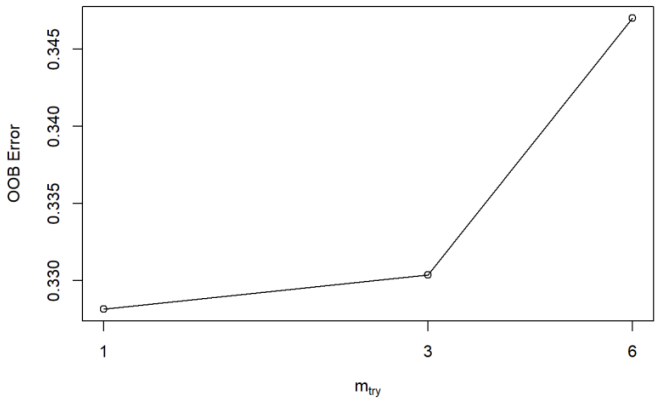
Step 1: Build a primary Random Forest model

```
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
```

Step 2: Find out the optimum value for ntree and mtry by plotting the following graphs and tune the model accordingly.






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

From the 1st graph, we can say that value of ntree = 1000 is appropriate as the OOB error estimate gets saturated and from the 2nd graph, we can observe that the OOB error estimate is lowest from mtry=1 and hence, this is the appropriate value of mtry. Thus, we will tune the model accordingly.

Step 3: Performing predictions and generating the confusion matrix

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
```

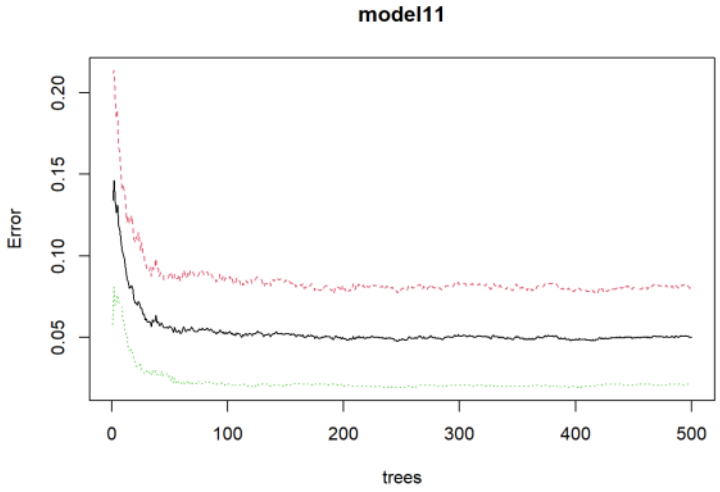
Random Forest with Both over and under sampling dataset:

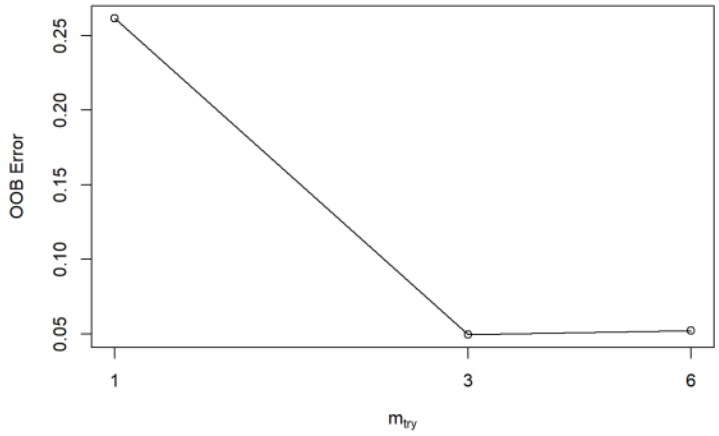
Step 1: Build a primary Random Forest model

```
set.seed(100)
model11<-randomForest(y~.,data=both,ntree=500,mtry=3)
model11
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = both, ntree = 500, mtry = 3)
##                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
```

Step 2: Find out the optimum value for ntree and mtry by plotting the following graphs and tune the model accordingly.






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

From the 1st graph, we can say that value of ntree = 500 is appropriate as the OOB error estimate gets saturated and from the 2nd graph, we can observe that the OOB error estimate is lowest from mtry=4 and hence, this is the appropriate value of mtry. Thus, we will tune the model accordingly.

Step 3: Performing predictions and generating the 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
```

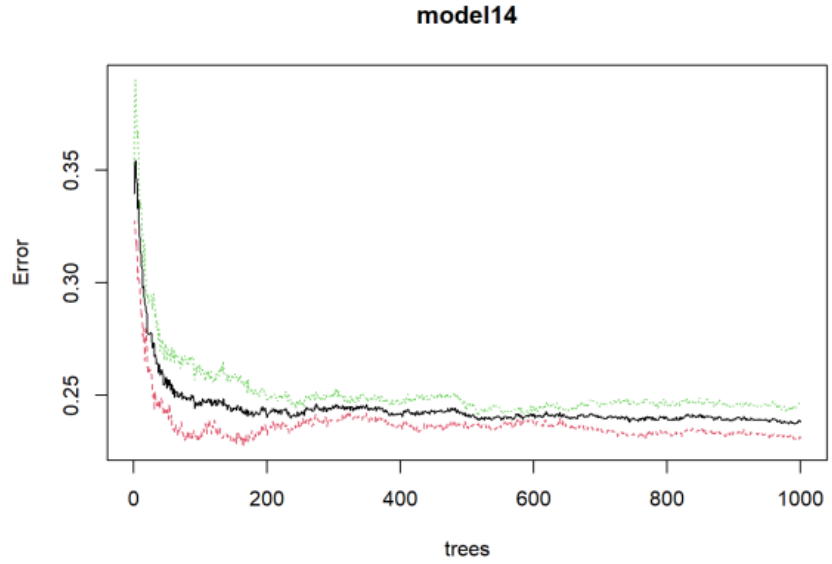
Random Forest with Synthetic dataset:

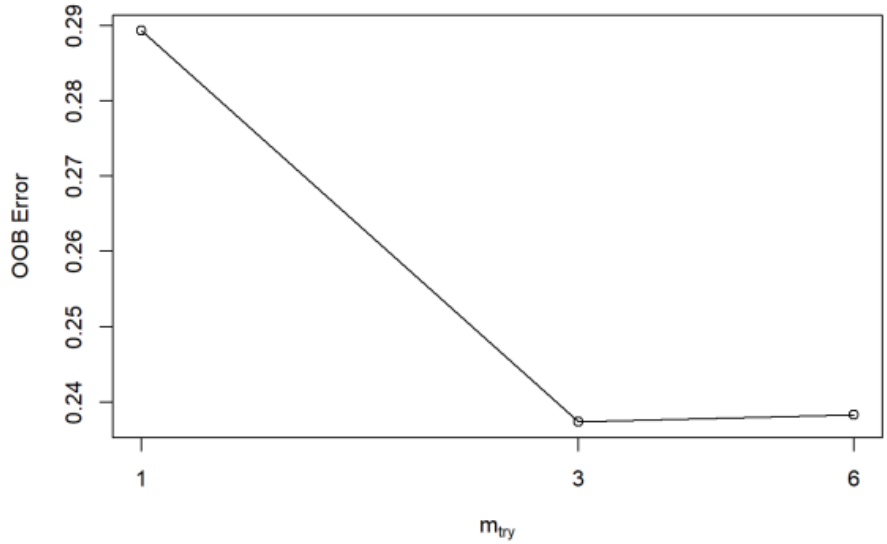
Step 1: Build a primary Random Forest model

```
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
```

Step 2: Find out the optimum value for ntree and mtry by plotting the following graphs and tune the model accordingly.






```
##       mtry  OOBError
## 1.OOB    1 0.2892874
## 3.OOB    3 0.2373761
## 6.OOB    6 0.2383200
```

From the 1st graph, we can say that value of ntree = 1000 is appropriate as the OOB error estimate gets saturated a lot and from the 2nd graph, we can observe that the OOB error estimate is lowest from mtry=3 and hence, this is the appropriate value of mtry. Thus, we will tune the model accordingly.

Step 3: Performing predictions and generating the 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
```

K Nearest Neighbours model with Oversampling dataset:

Step 1: Build a primary KNN model

```
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
```

Step2: Performing predictions and generating the confusion matrix

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
```

K Nearest Neighbours model with Under sampling dataset:

Step 1: Build a primary KNN model

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
```

Step2: Performing predictions and generating the confusion matrix

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
```

K Nearest Neighbours model with Both Over and Under sampling dataset:

Step 1: Build a primary KNN model

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
```

Step2: Performing predictions and generating the confusion matrix

```
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
```

K Nearest Neighbours model with Synthetic dataset:

Step 1: Build a primary KNN model

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
```

Step2: Performing predictions and generating the confusion matrix

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
```

Naïve Bayes Classifier with Oversampling dataset:

Step 1: Build a primary model using Naïve Bayes classifier

```
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
```

Step 2: Performing predictions and generating the confusion matrix

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

```
testprednb_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
```

Naïve Bayes Classifier with Under sampling dataset:

Step 1: Build a primary model using Naïve Bayes classifier

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
```

Step 2: Performing predictions and generating the confusion matrix

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
```

Naïve Bayes Classifier with Both Over and Under sampling dataset:

Step 1: Build a primary model using Naïve Bayes classifier

```
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
```

Step 2: Performing predictions and generating the 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
```

Naïve Bayes Classifier with Synthetic dataset:

Step 1: Build a primary model using Naïve Bayes classifier

```
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
```

Step 2: Performing predictions and generating the confusion matrix

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)
```

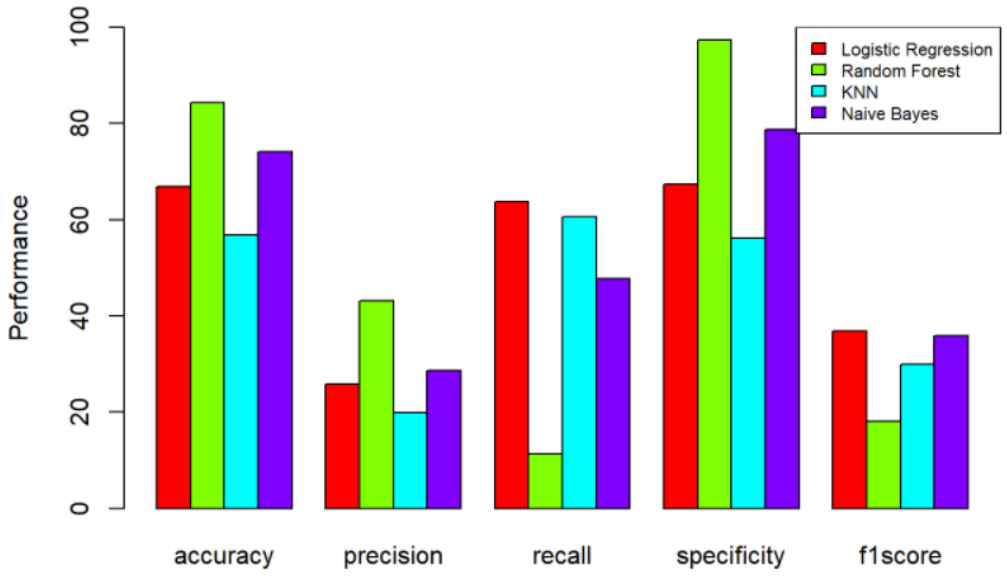
```
##       y_pred
## y_true   N   Y
##      N 888 190
##      Y 110  83
```

OUTCOMES

We will compare the results of all the classification techniques based on different measures taken to solve the issue of data imbalance.

Oversampling

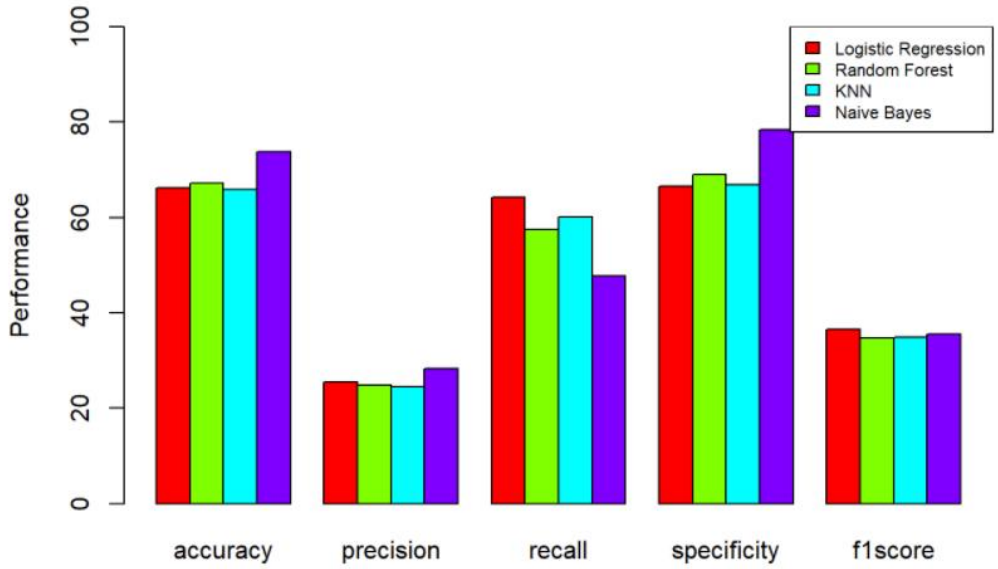
```
##                     accuracy precision   recall specificity  f1score
## Logistic Regression 66.79780  25.89474 63.73057    67.34694 36.82635
## Random Forest       84.26436  43.13725 11.39896    97.30983 18.03279
## KNN                 56.88434  19.86418 60.62176    56.21521 29.92327
## Naive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



We can observe that Logistic Regression, Naive Bayes and KNN performed a bit well under the technique of over sampling, but Random Forest didn't quite reach the standard set by the other three as it's precision and F Measure dropped quite a lot. If compared between Logistic Regression, Naive Bayes and KNN, **Logistic Regression** performed a bit better because it has very decent scores for all of the performance measures.

Under Sampling

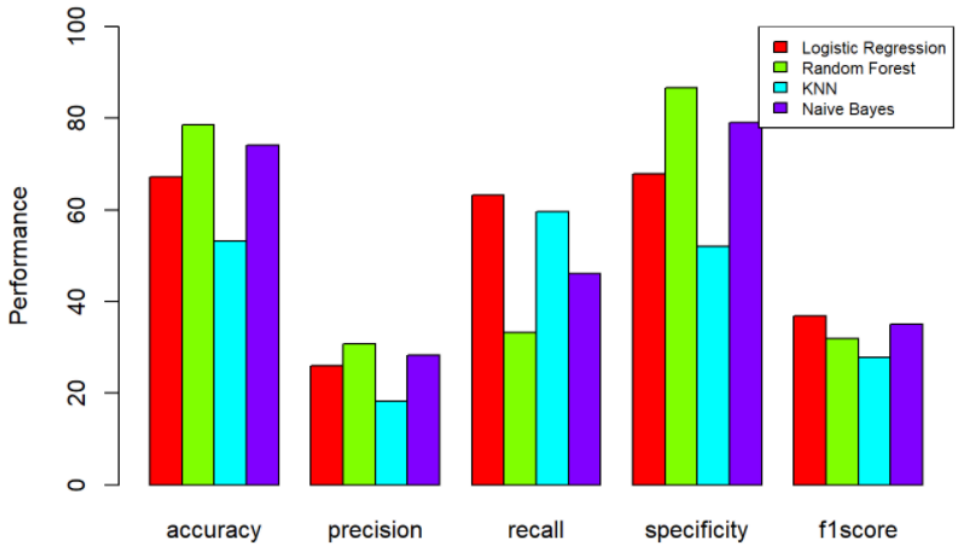
```
##                     accuracy precision   recall specificity  f1score
## Logistic Regression 66.16837  25.56701 64.24870    66.51206 36.57817
## Random Forest       67.19119  24.88789 57.51295    68.92393 34.74178
## KNN                 65.85366  24.52431 60.10363    66.88312 34.83483
## Naive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



Once again, **Logistic Regression** performs well as compared to the other classification techniques. It may not have a high accuracy and precision but the other scores such as recall, specificity scores are still above average and quite decent.

Both Over and Under Sampling

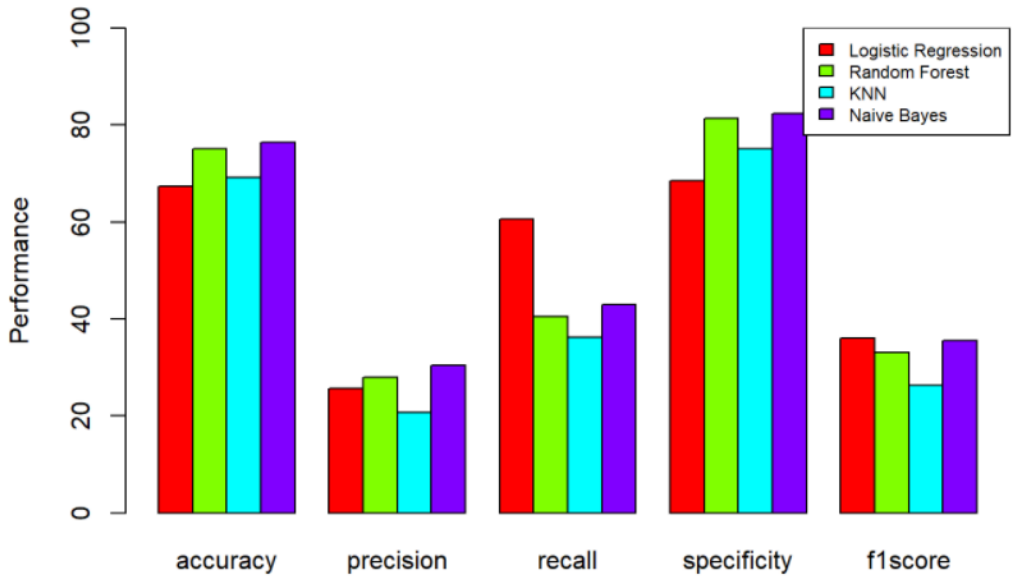
```
##                     accuracy precision   recall specificity  f1score
## Logistic Regression 67.11251  26.01279 63.21244    67.81076 36.85801
## Random Forest       78.52085  30.76923 33.16062    86.64193 31.92020
## KNN                 53.18647  18.19620 59.58549    52.04082 27.87879
## Naive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



Although Random Forest has better values in accuracy, but it's recall dropped a lot as compared to the other three. Considering the overall performance, we can conclude that, once again **Logistic Regression** has given the best predictions in the case of both over and under sampling technique as well.

Synthetic Dataset

```
##                     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
```



We can observe that Logistic Regression had very decent scores of accuracy, recall and specificity. Even it's F1 Score is quite similar to the other classification techniques although it's precision dropped quite a lot but it's still similar to the precision values of other techniques. Hence, we can state that Logistic Regression performed well under synthetic dataset as well.

Logistic Regression worked perfectly for the problem of binary classification. It excelled in all the scenarios of solving data imbalance done by different techniques.

RESOURCES

- **Kaggle** (source of data)
- **MS Excel** (data conversion)
- **RStudio** (classification models and testing)
- **Google and YouTube** (for concept and coding)

IMPORTANT LINKS

Google:

- https://www.geeksforgeeks.org/logistic-regression-in-r-programming/
- https://www.geeksforgeeks.org/random-forest-approach-in-r-programming/
- https://www.geeksforgeeks.org/classifying-data-using-support-vector-machinessvms-in-r/
- https://www.geeksforgeeks.org/naive-bayes-classifier-in-r-programming/

YouTube:

- https://www.youtube.com/watch?v=C4N3_XJJ-jU&t=805s
- https://www.youtube.com/watch?v=XycruVLySDg&t=8s
- https://www.youtube.com/watch?v=Z5WKQr4H4Xk
- https://www.youtube.com/watch?v=AVx7Wc1CQ7Y
- https://www.youtube.com/watch?v=6EXPYzbfLCE&t=807s
- https://www.youtube.com/watch?v=dJclNIN-TPo&t=1270s
- https://www.youtube.com/watch?v=HeTT73WxKIc
- https://www.youtube.com/watch?v=acFviblzijU
- https://www.youtube.com/watch?v=RKZoJVMr6CU&t=1544s
- https://www.youtube.com/watch?v=pS5gXENd3a4&t=524s
- https://www.youtube.com/watch?v=QkAmOb1AMrY&t=1013s
- https://www.youtube.com/watch?v=ueKqDlMxueE&t=731s
- https://www.youtube.com/watch?v=RLjSQdcg8AM
- https://www.youtube.com/watch?v=psHrcSacU9Y