GROUP PROJECT: Classification

PREDICTIVE ANALYTICS

PROF. TAO XU

STAT8031: MULTIVARIATE STATISTICS

Name - Suraj Mishra

Table of Contents

INTRODUCTION

➢ PROBLEM STATEMENT:

The Project Goal is to develop a classification model to predict the presence or absence of diabetes in individuals based on their demographic and a given set of clinical measurements available in the dataset.

In this group project, we will create five different classification models on the training dataset and will run predictions on the test dataset. Next, we will use these five models and compare their accuracies to determine which one is the best-suited classification model for our problem statement.

➢ ABOUT DATASET:

The dataset is collected from the National Institute of Diabetes and Digestive and Kidney Diseases. The dataset has a total of 9 features and around 768 rows of data each representing a diabetes patient.

diabetes_classification.csv

Below is the summary of all features in the dataset :

Feature	**Type of Feature**	**Description**
Pregnancies	Quantitative	Number of pregnancies the patient has had so far.
Glucose	Quantitative	Glucose content in milligrams per deciliter
BloodPressure	Quantitative	Blood Pressure in mmHg (millimeter mercury)
SkinThickness	Quantitative	Measure of thickness of the skin

Insulin	Quantitative	Level of Insulin in the blood sample of the patient.
BMI	Quantitative	Body Mass Index and can take float values.
DiabetesPedigreeFunction	Quantitative	The likelihood index that depends upon the patient’s diabetic family history. This can take float values as well.
Age	Quantitative	Age of the patient in years
Outcome	**Qualitative**	This is the response variable in this project where we predict 0 (“No”) or 1 (“Yes”) based on the clinical measurements described above.

Response Variable	Outcome
Predictors	Rest of the Variables in the dataset

DATASET SPLIT & FACTORIZATION

Data splitting is necessary to evaluate the performance of the model on unseen data and prevent overfitting. Hence, the dataset is split into two portions called “**train**” and “**test**” portions

In this project, the dataset is split randomly using R, with 80% data in the training set (614 rows) and 20% data (154 rows) in the testing set. Split of the dataset is done as below :

- This partition is done using the initial_split() in R which allows to provide a proportion value for the training set that should extracted in the split.
- Initial_split() uses below criteria :
  Proportion: 0.8
  Strata =Age

We used stratified sampling here by “Age” to ensure that the number of data points in the training data is equivalent to Outcome proportions in the dataset.

Since we are doing a classification case study, sampling here is conducted separately with each of the classes under Outcome (as 0 and 1).

Factorization

For the sake of simplicity, we factorized the categories under our response variable “ Outcome” using factor() in R with below logic :

Outcome=0 is factorized as “No” which means the patient is **not Diabetic**
Outcome=1 is factorized as “Yes” which represents that the patient is **Diabetic**.

(***Refer factorization codes in the R file attached with the project)

METHODOLOGIES

1. <u>K-Nearest Neigbours Classification Model</u>

- **Overview:**

The k-nearest neighbors (KNN) algorithm is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.

  - It is widely disposable in real-life scenarios since it is non-parametric, meaning it does not make any underlying assumptions about the distribution of data.

  - It works by finding the K nearest neighbors to a given data point based on a distance metric, such as Euclidean distance. The class or value of the data point is then determined by the majority vote or average of the K neighbors.

- **Implementation details**:

  - <u>Pre-processing steps</u>:

    We use center and scale to preprocess the KNN model to estimate the scale and location of the predictors.

    This is done because KNN is a distance-based algorithm and hence if data is not scaled, the predictors with larger scale might dominate the distance calculations and thus would result in incorrect classification.

  - <u>Tuning hyperparameter</u> "k" in KNN using tuneGrid:

    In this model, we have chosen the value of k between 1 and 5 and **used a tuning grid to tune the value k between 1 and 5**. This is done to resample the results across each of the values of k between 1 and 5.

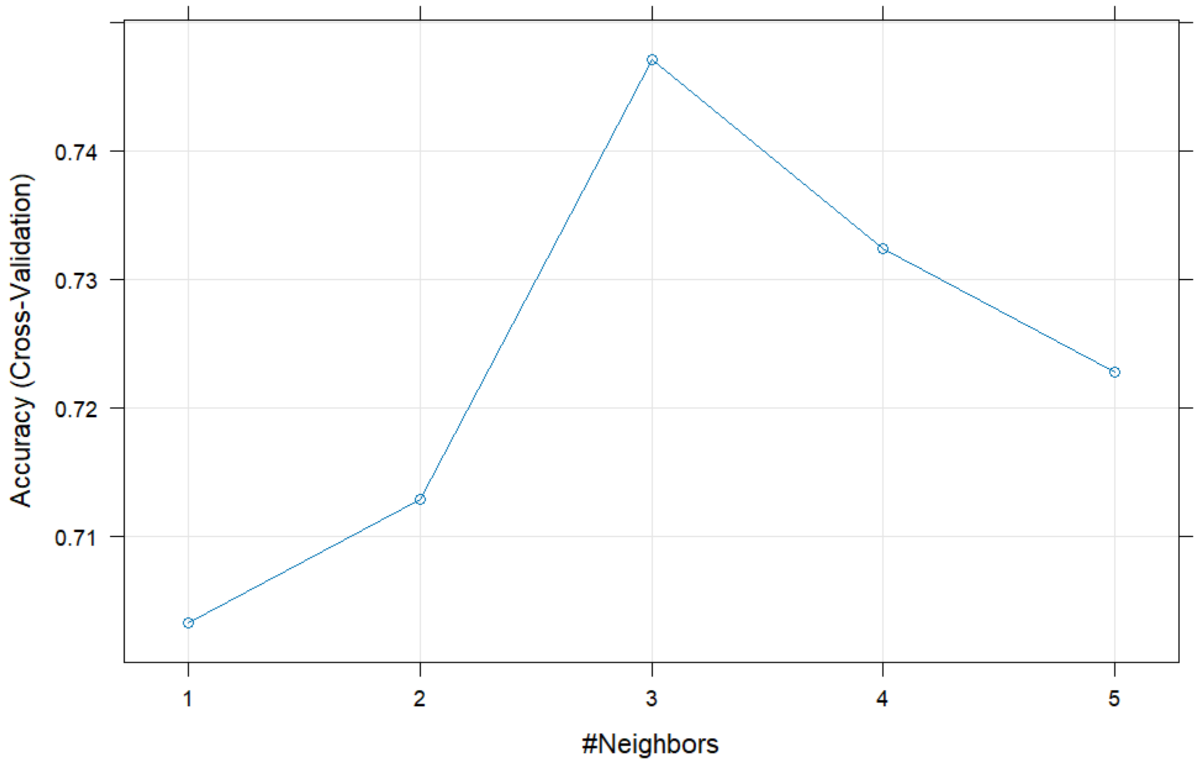
- Resampling during model selection:

  We use the iterative method of k-fold cross-validation for resampling using the trainControl(), where the model is trained and evaluated k times. In this model, we have used **10-fold cross-validation** where each iteration keeps one partition for testing and the remaining 9 partitions for training the model.

- **Model Training**:
  Once have set up our hyperparameters and tune Grid, we now train the KNN model using the train().

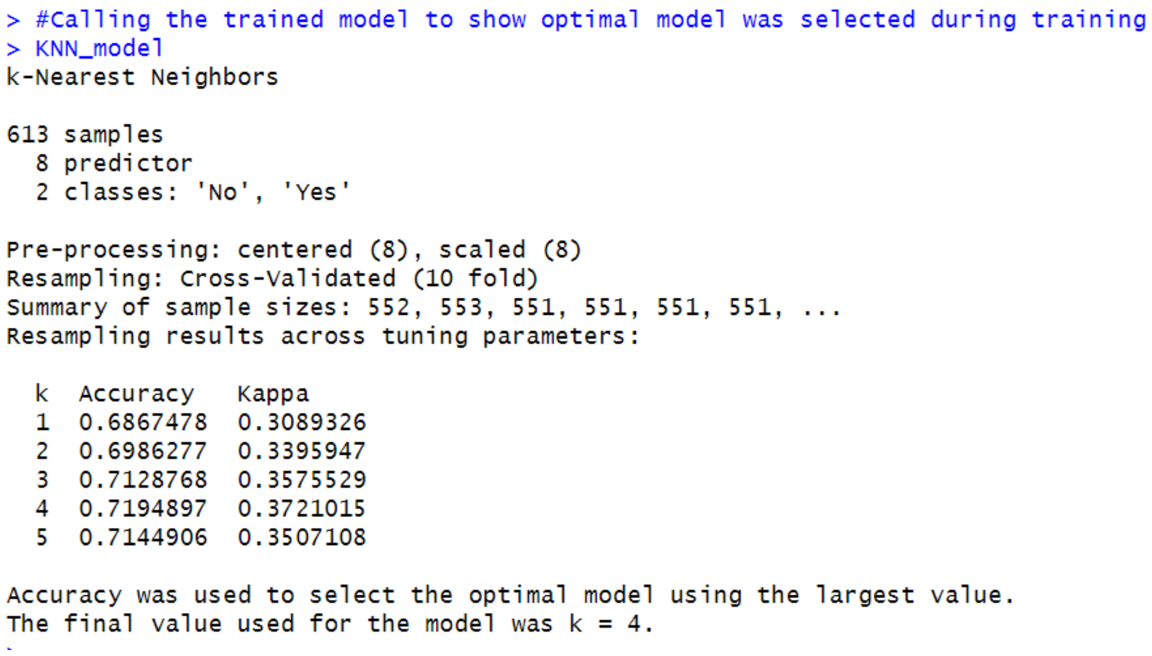
  The training will result in the best model selection with the best accuracy out of the 5 chosen values of "k". (******the codes are shared in the R file attached and shared along with the project***)

  Below we have the plot we generated for the trained model which shows variation of accuracy with all values of k that we have considered here:



In practice, the optimal value of k is usually found in the square root of N, where N =10 is the number of Cross-validation samples we took during the resampling process in the KNN model.

Based on the trained model results, **k=4 was selected as the best model** as shown below :



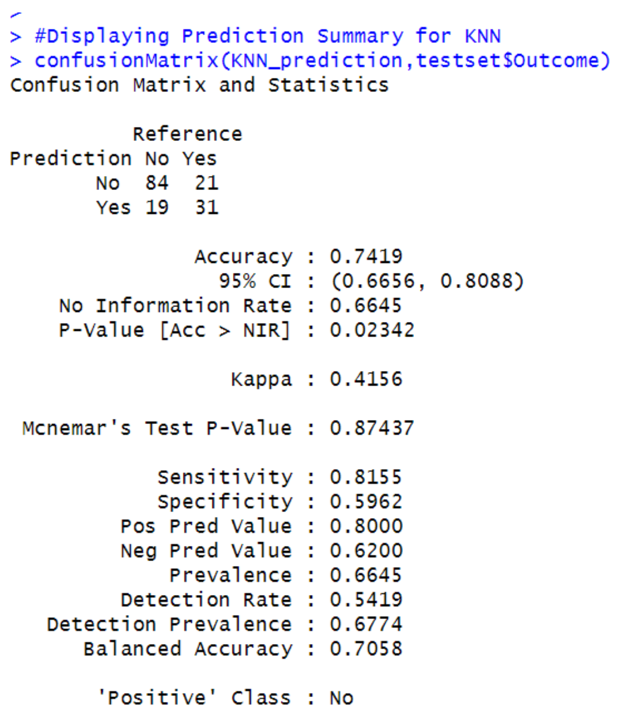
```
> #Calling the trained model to show optimal model was selected during training
> KNN_model
k-Nearest Neighbors

613 samples
  8 predictor
  2 classes: 'No', 'Yes'

Pre-processing: centered (8), scaled (8)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 552, 553, 551, 551, 551, 551, ...
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  1  0.6867478  0.3089326
  2  0.6986277  0.3395947
  3  0.7128768  0.3575529
  4  0.7194897  0.3721015
  5  0.7144906  0.3507108

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 4.
```

- **Model Evaluation**

  After the Model training, we ran predictions of the KNN model on the test data to generate predictions on the Outcome response variable.

  The predictions were then plotted using the Confusion matrix that displayed the following results :

  (**Screenshot on the next page**)



```
> #Displaying Prediction Summary for KNN
> confusionMatrix(KNN_prediction,testset$Outcome)
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No  84  21
       Yes 19  31

               Accuracy : 0.7419
                 95% CI : (0.6656, 0.8088)
    No Information Rate : 0.6645
    P-Value [Acc > NIR] : 0.02342

                  Kappa : 0.4156

 Mcnemar's Test P-Value : 0.87437

            Sensitivity : 0.8155
            Specificity : 0.5962
         Pos Pred Value : 0.8000
         Neg Pred Value : 0.6200
             Prevalence : 0.6645
         Detection Rate : 0.5419
   Detection Prevalence : 0.6774
      Balanced Accuracy : 0.7058

       'Positive' Class : No
```

2. Decision Tree Classification Model

- **Overview**:

  A classification decision tree is used to predict a qualitative response based on a given set of features. For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs. In keeping with the tree analogy, the categories within the response variable are referred to as **terminal nodes** or leaves of a tree. The points along the tree where the predictor space is split are referred to as **internal nodes**. We refer to the node segments of the trees that connect the nodes as **branches**.

  In interpreting the results of a classification tree, we are often interested not only in the class prediction corresponding to a particular terminal node region but also in the class proportions among the training observations that fall into that region.

- **Implementation Details for Decision Tree Model:**

- Decision Tree Complexity: The tree's complexity was automatically determined by the rpart package based on default hyperparameters such as minsplit, minbucket, cp, and maxdepth.

- Feature selection: The decision tree model uses the GINI index to select best features that can be used for classification.

  The Gini Index is a statistic that is utilized to determine the node purity at every split done at every node of the tree.

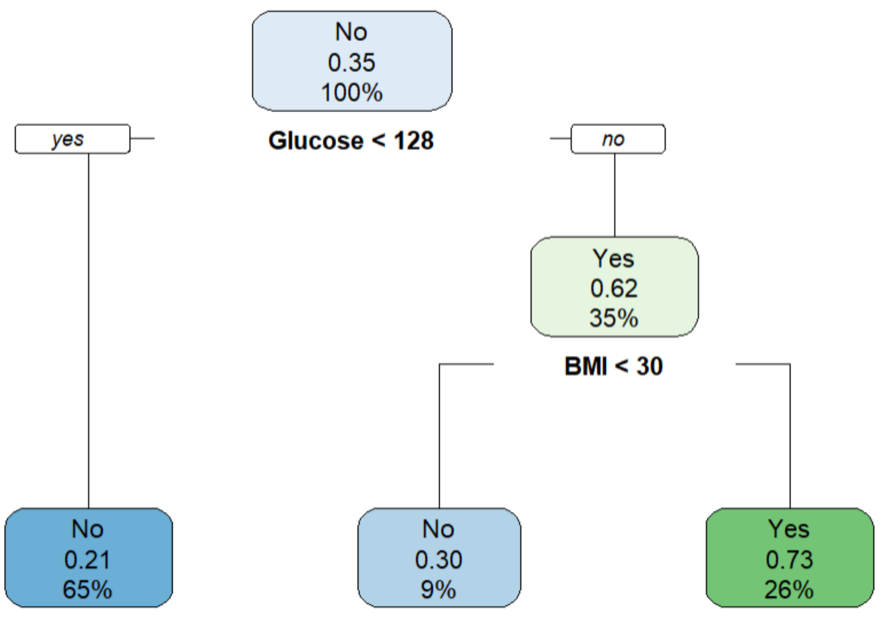
- Stopping conditions, like maximum tree depth or minimum node size, were applied to prevent overfitting and enhance the model's generalization.

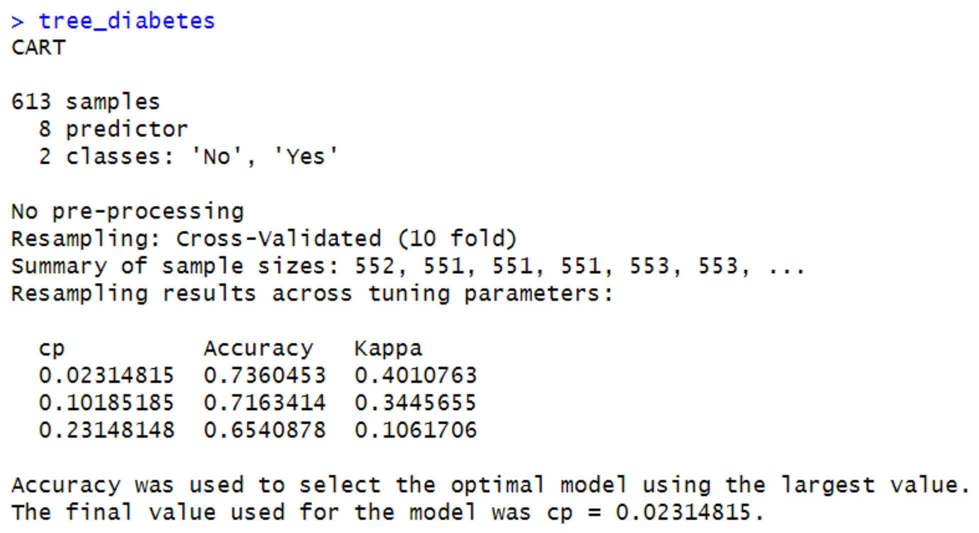
- **Model Training:**

  We used the rpart method is training of the decision tree model on the training dataset. Further, we also used the 10-fold cross validation method for resampling as we did in the other classification models in this project.

  The train() method uses one of the default hyperparameters as "cp" to improve the accuracy of the decision tree by controlling the size of the decision tree and avoiding overfitting. This method is then used in pruning the tree, which involves removing the branches that have weak predictive power.

  Based on the trained model we get below decision tree with cp value =0.02314815





```
> tree_diabetes
CART

613 samples
  8 predictor
  2 classes: 'No', 'Yes'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 552, 551, 551, 551, 553, 553, ...
Resampling results across tuning parameters:

  cp          Accuracy   Kappa
  0.02314815  0.7360453  0.4010763
  0.10185185  0.7163414  0.3445655
  0.23148148  0.6540878  0.1061706

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.02314815.
```

In the trained decision tree model, the best predictor variables are evaluated as :

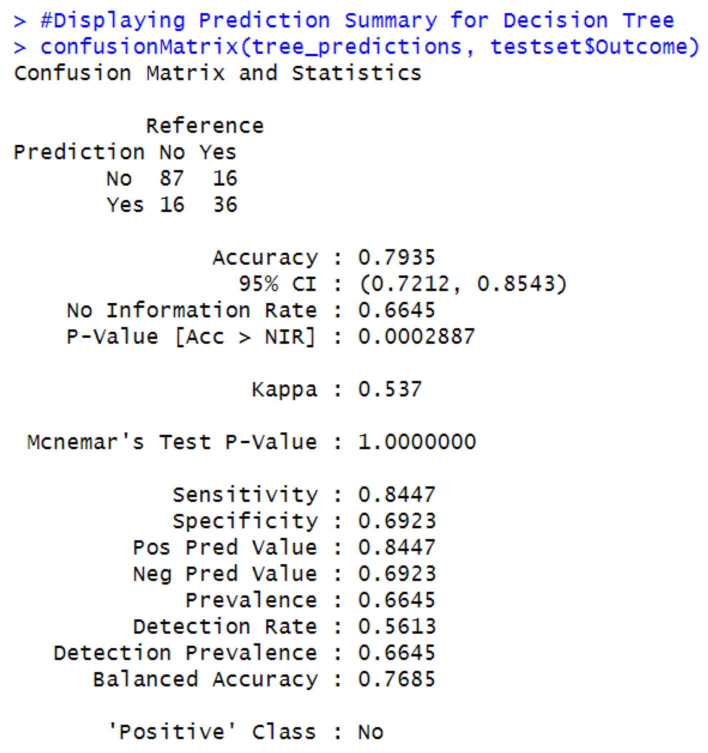
"**Glucose Level**" < 128 as the initial split which if True classifies the patient to Outcome as "No" (i.e, not diabetic).
Further, if this is false, the tree is split on the next best predictor which is "**BMI**" < 30 which if True classifies the patient to Outcome as "No" (i.e, not diabetic) else outcome as "Yes" (i.e., is diabetic).

- **Model Evaluation**

After the Model training, we ran predictions of the decision tree model on the test data to generate predictions on the Outcome response variable.

The predictions were then plotted using the Confusion matrix that displayed the following results :

(**Screenshot on the next page**)



```
> #Displaying Prediction Summary for Decision Tree
> confusionMatrix(tree_predictions, testset$Outcome)
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No  87  16
       Yes 16  36

               Accuracy : 0.7935
                 95% CI : (0.7212, 0.8543)
    No Information Rate : 0.6645
    P-Value [Acc > NIR] : 0.0002887

                  Kappa : 0.537

 Mcnemar's Test P-Value : 1.0000000

            Sensitivity : 0.8447
            Specificity : 0.6923
         Pos Pred Value : 0.8447
         Neg Pred Value : 0.6923
             Prevalence : 0.6645
         Detection Rate : 0.5613
   Detection Prevalence : 0.6645
      Balanced Accuracy : 0.7685

       'Positive' Class : No
```

3. LOGISTIC CLASSIFICATION MODEL

- **Overview**:

It's a classification algorithm, that is used where the response variable is categorical. The idea of Logistic Regression is to find a relationship between features and probability of particular outcome. The dependent variable is dichotomous in nature, i.e. there could only be two possible classes. This type of a problem is referred to as Binomial Logistic Regression, where the response variable has two values 0 and 1 or "true" and "false" or "yes" and "no".

Hence, the Logistic model is a predictive analysis algorithm and based on the concept of probability.

- **Implementation Details for Logistic Model:**

  - method**:**
    In logistic model , we have used method as "glm" as logistic regression belongs to the class of generalized linear models. This method helps to train the data by creating a logit model of the response variable. glm method is available to us by the caret library in R.

- <u>family</u>:
  we pass this parameter in the train function for a logistic model which tells R which type of regression we want to run. Since, we are using classification of Outcome response , we specify family as “binomial” and hence R trains it as a binaomial logistic model.

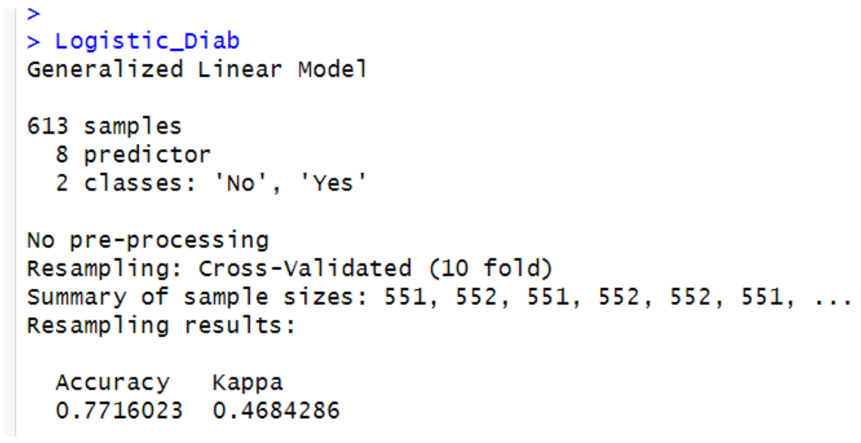
- <u>Resampling during model selection</u>:
  Similar to what we used in the KNN classification trained model, we use the 10-fold cross-validation in the logistic classification model as well. Hence, we pass the “cv” method with a value of 10 in the trainControl().

- <u>Maximum likelihood function</u>:
  Since logistic classification is a probability-based classification, it uses an **iterative maximum likelihood function** that seeks to find estimates of the logistic regression coefficients such that the predicted probability of a patient’s Outcome value is as close as possible to the patient’s observed Outcome value.
  Also, the logistic model in R works with the Fisher method which fits the model by iteratively re-weighting the least squares.

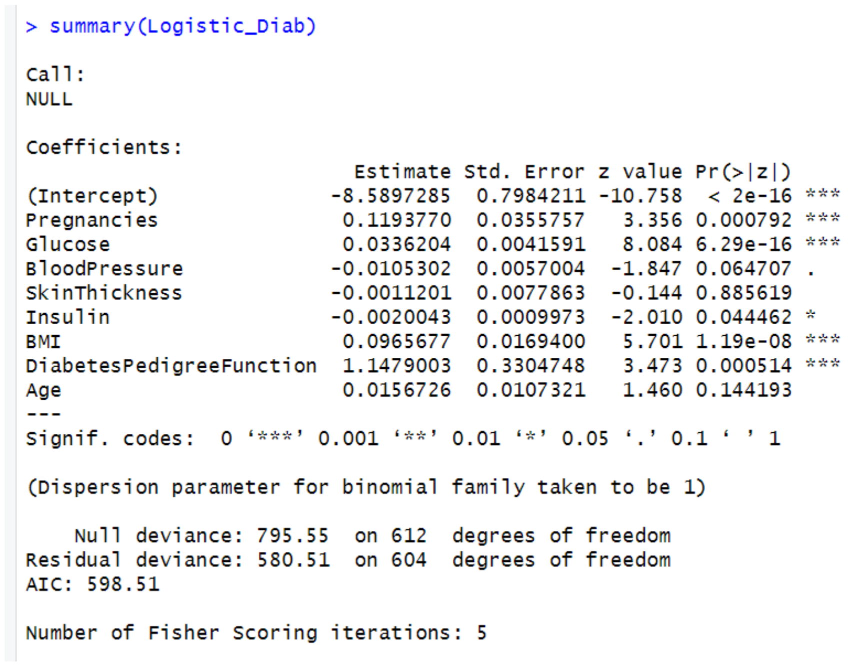
- **Model Training**

  Once we have passed all the parameters as mentioned into the train(), we fetch the results of the logistic model trained using the training set.

  Below are results from training the logistic model:



```
>
> Logistic_Diab
Generalized Linear Model

613 samples
  8 predictor
  2 classes: 'No', 'Yes'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 551, 552, 551, 552, 552, 551, ...
Resampling results:

  Accuracy   Kappa
  0.7716023  0.4684286
```

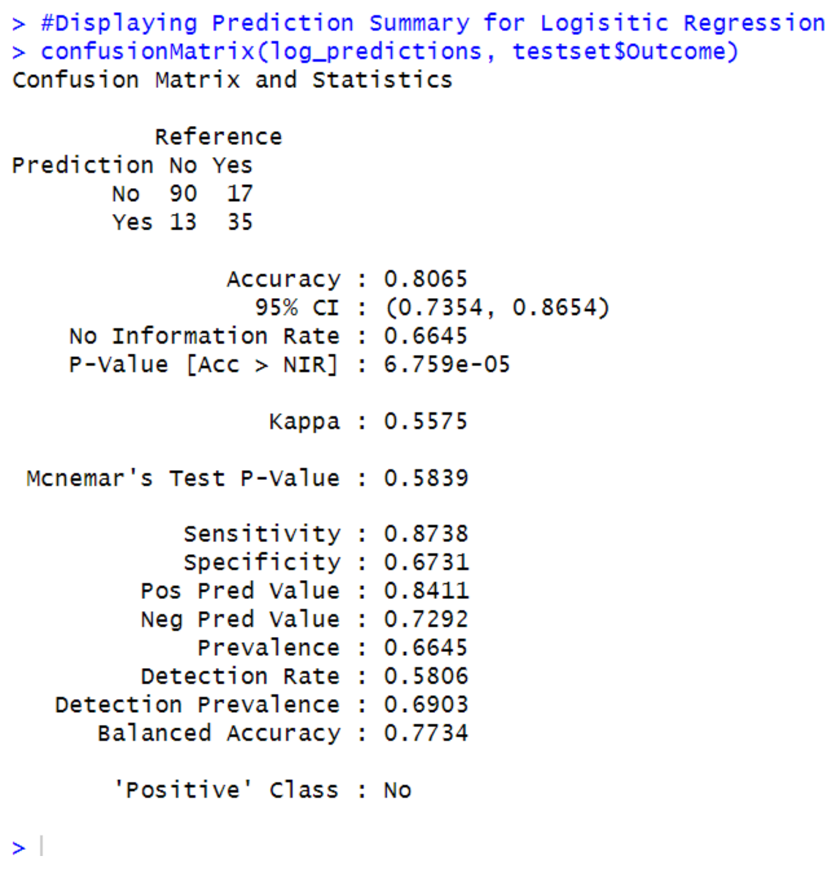

```
> summary(Logistic_Diab)

Call:
NULL

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)    
(Intercept)             -8.5897285  0.7984211 -10.758  < 2e-16 ***
Pregnancies              0.1193770  0.0355757   3.356 0.000792 ***
Glucose                  0.0336204  0.0041591   8.084 6.29e-16 ***
BloodPressure           -0.0105302  0.0057004  -1.847 0.064707 .  
SkinThickness           -0.0011201  0.0077863  -0.144 0.885619    
Insulin                 -0.0020043  0.0009973  -2.010 0.044462 *  
BMI                      0.0965677  0.0169400   5.701 1.19e-08 ***
DiabetesPedigreeFunction 1.1479003  0.3304748   3.473 0.000514 ***
Age                      0.0156726  0.0107321   1.460 0.144193    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 795.55  on 612  degrees of freedom
Residual deviance: 580.51  on 604  degrees of freedom
AIC: 598.51

Number of Fisher Scoring iterations: 5
```

The above output shows beta coefficients for logistic model for each of the predictors. By far, pregancies, glucose, BMI and Diabetes Pedigree Function are the most significant predictors in our trained model.

- **Model Evaluation**

  After the Model training, we ran predictions of the logistic classification on the test data to generate predictions on the Outcome response variable.

  The predictions were then plotted using the Confusion matrix that displayed the following results :



```
> #Displaying Prediction Summary for Logisitic Regression
> confusionMatrix(log_predictions, testset$Outcome)
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No  90  17
       Yes 13  35

               Accuracy : 0.8065
                 95% CI : (0.7354, 0.8654)
    No Information Rate : 0.6645
    P-Value [Acc > NIR] : 6.759e-05

                  Kappa : 0.5575

 Mcnemar's Test P-Value : 0.5839

            Sensitivity : 0.8738
            Specificity : 0.6731
         Pos Pred Value : 0.8411
         Neg Pred Value : 0.7292
             Prevalence : 0.6645
         Detection Rate : 0.5806
   Detection Prevalence : 0.6903
      Balanced Accuracy : 0.7734

       'Positive' Class : No

> |
```

4. Random Forest Classification Model

- **Overview**:

The Random Forest algorithm is an ensemble learning technique utilized for classification and regression tasks. It leverages multiple decision trees to enhance prediction accuracy and robustness. Random Forest employs bagging, where each tree is trained on a subset of the training data. The algorithm's strength lies in its ability to handle complex datasets and mitigate overfitting, making it a valuable tool for various predictive tasks in machine learning.

- **Implementation details**:

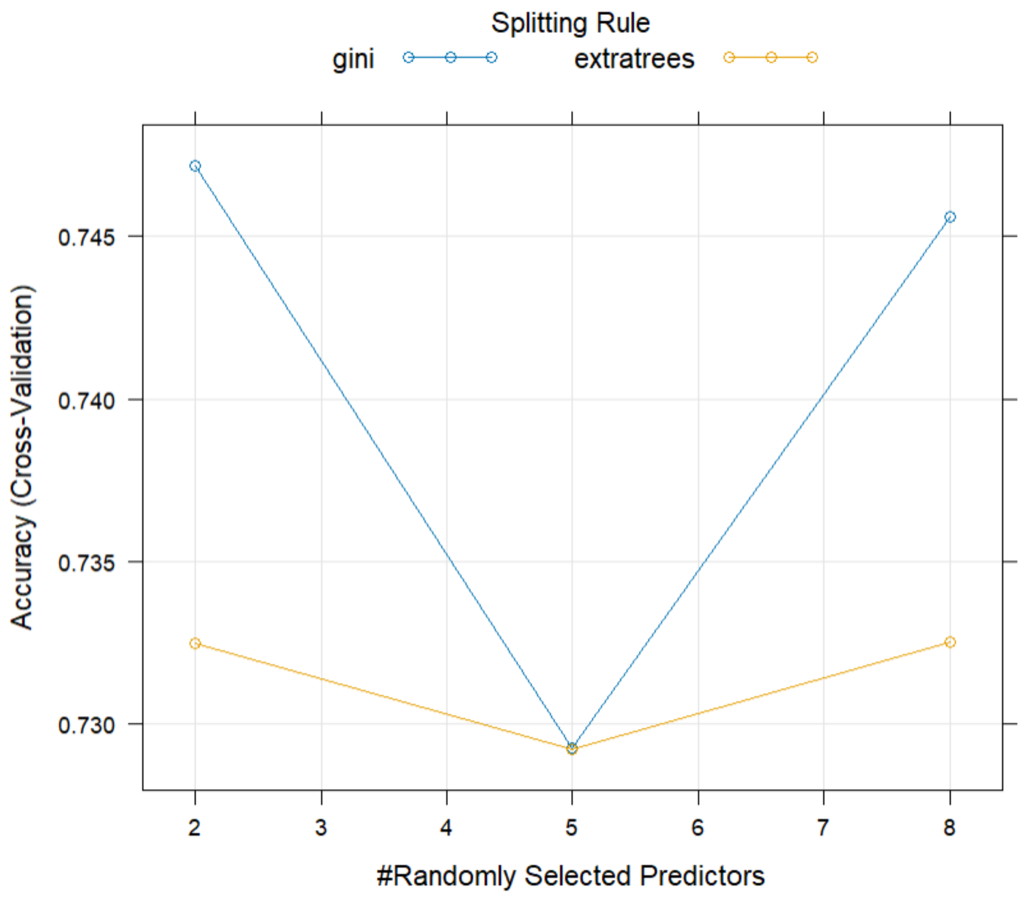
  - Feature Selection - In our Random Forest model, we have used all features except the outcome variable as predictors. This allows us to utilize the dataset's full information while excluding the target variable, ensuring accurate predictions.

  - Hyperparameter – In our Random Forest model, the number of trees ('num.trees') was set to 100. This hyperparameter determines the quantity of decision trees generated in the forest. By adjusting the number of trees, we control the model's complexity and predictive accuracy. We tried it implementation through 150, 200 number of trees initially. However, the model's performance and its accuracy were good with 100 trees.

- Resampling Method during model selection– For evaluating the performance of our Random Forest model, we employed cross-validation as the resampling technique. Specifically, we conducted 5-fold cross-validation (number = 5), where the dataset was divided into five subsets for training and testing. This approach enables a thorough assessment of the model's ability to generalize to unseen data and mitigates concerns about overfitting.

- Model Training: For training our Random Forest model, we utilized the train() function. During this process, we optimized the model's hyperparameters to select the best configuration. The number of trees in the forest was set to 100, and variable importance was determined based on impurity reduction.

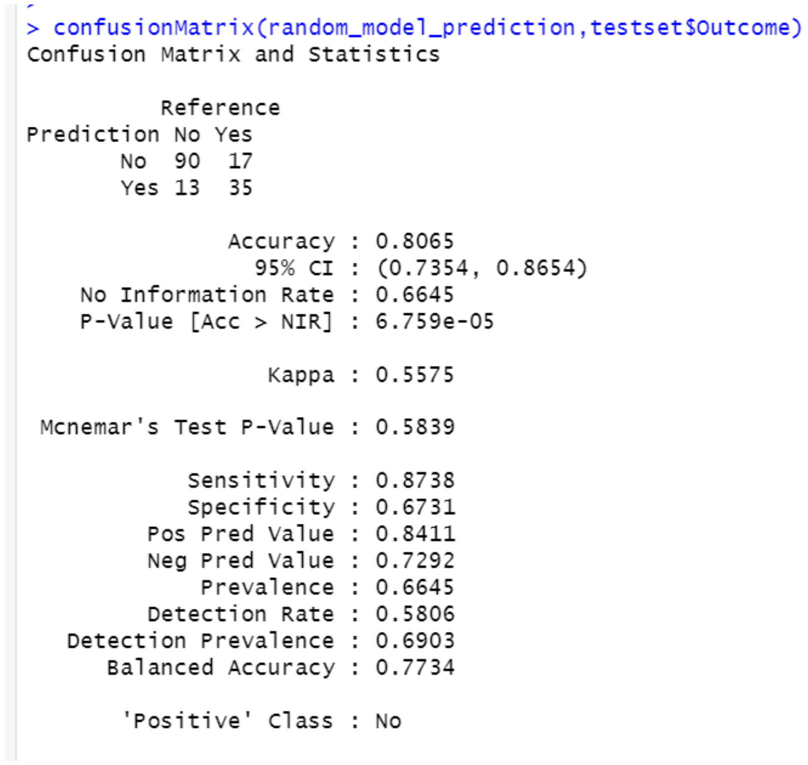
Below we have the plot we generated for the trained model which shows variation of accuracy with randomly selected predictors based on two criteria Gini impurity and extratrees:



- **Model Evaluation**

  After the Model training, we ran predictions of the random forest classification model on the test data to generate predictions on the Outcome response variable.

  The predictions were then plotted using the Confusion matrix that displayed the following results :



```
> confusionMatrix(random_model_prediction,testset$Outcome)
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No  90  17
       Yes 13  35

               Accuracy : 0.8065
                 95% CI : (0.7354, 0.8654)
    No Information Rate : 0.6645
    P-Value [Acc > NIR] : 6.759e-05

                  Kappa : 0.5575

 Mcnemar's Test P-Value : 0.5839

            Sensitivity : 0.8738
            Specificity : 0.6731
         Pos Pred Value : 0.8411
         Neg Pred Value : 0.7292
             Prevalence : 0.6645
         Detection Rate : 0.5806
   Detection Prevalence : 0.6903
      Balanced Accuracy : 0.7734

       'Positive' Class : No
```

5. <u>Linear Discriminant Analysis</u>

- **Overview**:
  Linear discriminant analysis (LDA) is a method employed in supervised machine learning for addressing multi-class classification tasks. LDA works by reducing the dimensionality of data to effectively distinguish between multiple classes based on multiple features. This approach plays a crucial role in data science by enhancing the performance of machine learning models.

    - The linear discriminant functions produced by LDA are interpretable and can provide insights into the importance of different features for classification.

- LDA is less prone to overfitting compared to more complex models, especially when the number of samples is small relative to the number of features.

- **Implementation Details for Linear discriminant Analysis Model:**

  - Prior Probabilities -

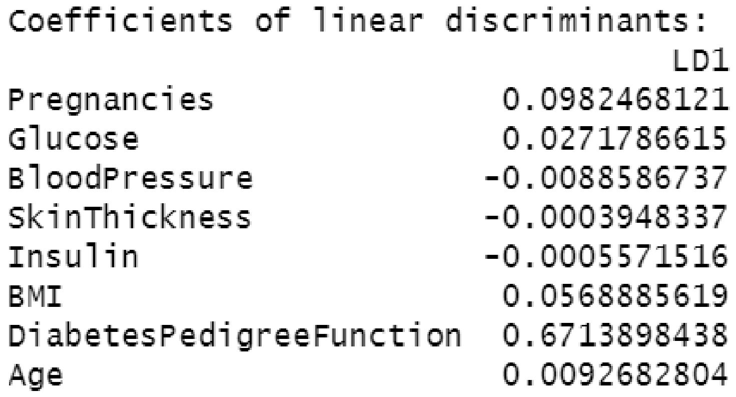
Prior probabilities represent the likelihood of an observation belonging to a specific class even before any measurements or features are considered. These probabilities are denoted by π (pi) for each class.

For our analysis with two classes, "No" and "Yes", we have established the following prior probabilities based on P(No) = 0.6525 (or πNo = 0.6525): This indicates a 65.25% chance an observation belongs to the "No" class.

P(Yes) = 0.3475 (or πYes = 0.3475): This indicates a 34.75% chance an observation belongs to the "Yes" class.

  - Coefficients of linear discriminants**:**

  LDA uses coefficients to determine how much each feature contributes to separating the classes ("No" and "Yes"). The provided table shows these coefficients for the first discriminant function (LD1).



```
Coefficients of linear discriminants:
                                  LD1
Pregnancies                0.0982468121
Glucose                    0.0271786615
BloodPressure             -0.0088586737
SkinThickness             -0.0003948337
Insulin                   -0.0005571516
BMI                        0.0568885619
DiabetesPedigreeFunction   0.6713898438
Age                        0.0092682804
```

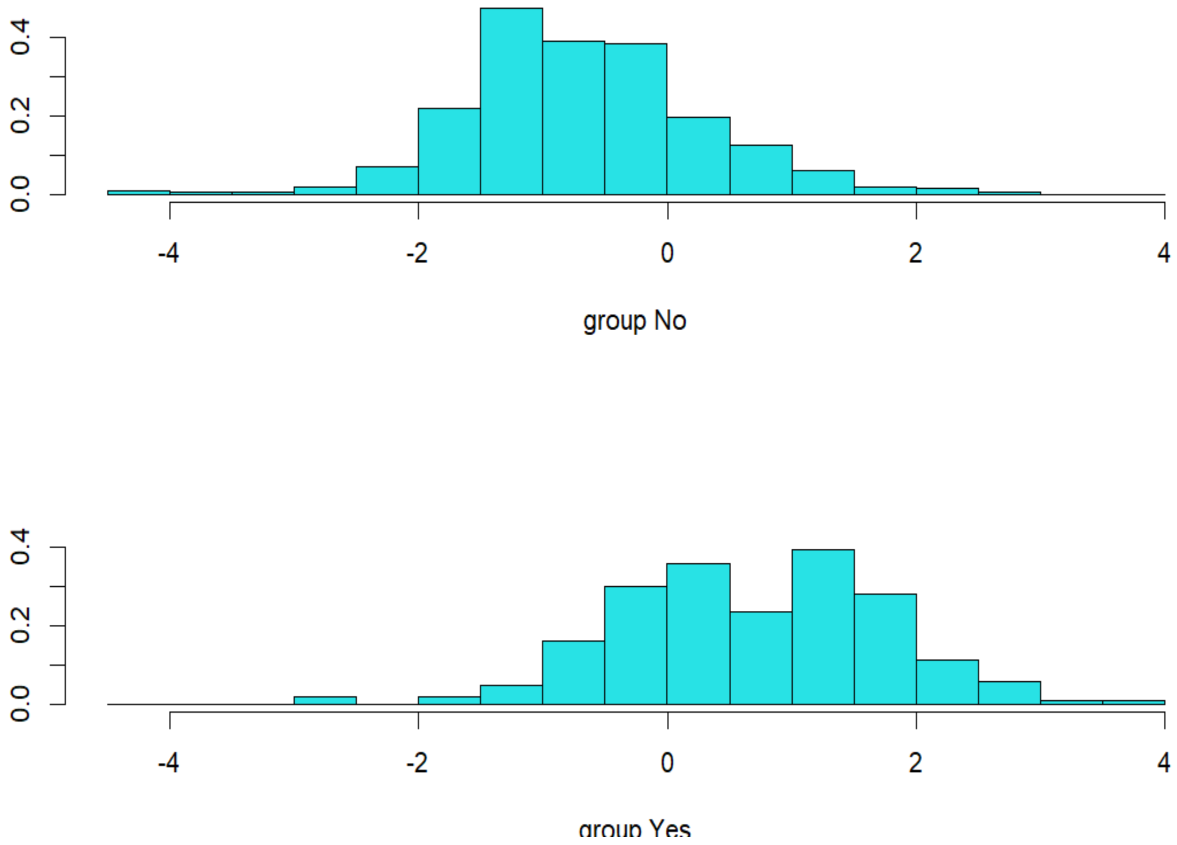
Higher absolute values indicate a stronger influence on class separation. Analyze class labels to understand the direction of influence (positive or negative coefficients).

For example, DiabetesPedigreeFunction has the strongest influence in LD1, but further analysis is needed to determine its specific effect on class distinction ("positive" or "negative").

These coefficients help identify the most important features for differentiating the classes in LD1.

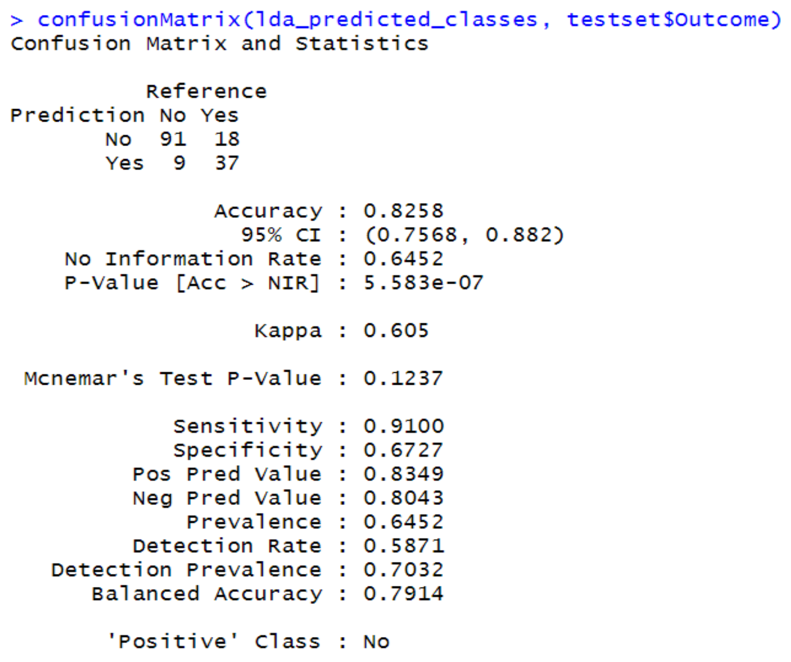
- **Model Training:** Applied Linear Discriminant Analysis lda() to build a statistical model that learns the patterns in the data and can predict the outcome variable.

  Below is the plot of LD1 which is classifying the group "yes" and "now" as per the normal distribution.



- **Model Evaluation:**
  After the Model training, we ran predictions of the LDA model on the test data to generate predictions on the Outcome response variable.

  The predictions were then plotted using the Confusion matrix that displayed the following results:



```
> confusionMatrix(lda_predicted_classes, testset$Outcome)
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No  91  18
       Yes  9  37
                                          
               Accuracy : 0.8258          
                 95% CI : (0.7568, 0.882) 
    No Information Rate : 0.6452          
    P-Value [Acc > NIR] : 5.583e-07       
                                          
                  Kappa : 0.605           
                                          
 Mcnemar's Test P-Value : 0.1237          
                                          
            Sensitivity : 0.9100          
            Specificity : 0.6727          
         Pos Pred Value : 0.8349          
         Neg Pred Value : 0.8043          
             Prevalence : 0.6452          
         Detection Rate : 0.5871          
   Detection Prevalence : 0.7032          
      Balanced Accuracy : 0.7914          
                                          
       'Positive' Class : No              
```

PERFORMANCE COMPARISON

Based on the predictions obtained from all the classification models , below is the performance summary of all classification models:

Model Name	Accuracy	Sensitivity	Specificity
KNN Classification	74.19%	81.55%	59.62%
Decision Tree	79.35%	84.47%	69.23%
Logistic Classification	80.65%	87.38%	67.31%
Random Forest	80.65%	87.38%	67.31%
Linear Discriminant Analysis	82.58%	91%	67.27%

<u>Interpretation of the Performance metrics</u> :

In the evaluation of the performance metrics we use below terms :

TP (true positives): number of observations that were classified "Outcome" as Yes in test dataset and were also "Yes" in the trained dataset.

TN(true negatives): number of observations that were classified "Outcome" as No in test dataset and were also "No" in the trained dataset

FP (false positives): number of observations that were classified "Outcome" as Yes in test dataset but were "No" in the trained dataset.

FN (false negatives): number of observations that were classified "Outcome" as No in test dataset but were "Yes" in the trained dataset.

Based on the above terminology we find below parameters to compare the performance of all the models :

Accuracy = (**TP** + **TN**) / (**TP** + **TN** + **FP** + **FN**)

Sensitivity (true positive rate) = **TP** / (**TP** + **FN**)

Specificity (true negative rate) = **TN** / (**TN** + **FP**)

CONCLUSION

As per the performance comparison done above for all the five classification algorithms, we find that the performance of **Linear Discriminant Analysis** has the best accuracy of 82.58% which outperforms all other classification modes.