IDENTIFICATION FOR THE PRESENCE OF CARDIAC ARRHYTHMIA BY CREATING AN OPTIMAL PREDICTIVE MODEL BASED ON DATA COLLECTED FOR 452 PATIENTS

Sl. No.	*Last, First name*	*Percentage contribution*
1	Alvi, Sayeed Akhtar	20
2	Dhal, Sambandh Bhusan	20
3	Gupta, Saksham	20
4	Kohle, Khushboo Vilas	20
5	Parthasarathy, Sreenivas	20

Submitted to:
Dr. Na Zou,
Instructional Assistant Professor,
Dept. of Industrial and Systems Engineering
Texas A&M University

Submitted on:
May 01st 2018

Table of Contents

ABSTRACT 1
INTRODUCTION 1
PROJECT APPROACH 2
DATA DESCRIPTION 2
THE CHALLENGE 3
DATA PRE-PROCESSING 3
IMPLEMENTATION DETAILS AND FEATURE SELECTION 3-5
FEATURE SELECTION METHODS 5-9
METHODOLOGY 9-22
COMPARISON OF DIFFERENT METHODS 23
EXECUTIVE SUMMARY 23
CONCLUSION 23
FUTURE WORKS 23
REFERENCES 24
APPENDIX 25

Table of figures

Figure 1: Project flow diagram 2
Figure 2: Glmnet Equation 3
Figure 3: Before and After applying PCA technique to a sample data set 5
Figure 4: LASSO equation minimization 5
Figure 5: Logistical regression transformation equation and corresponding probability 6
Figure 6: Normal distribution and density function of predictor in a class 6
Figure 7: QDA equation form 7
Figure 8: KNN depiction and class assignment 7
Figure 9: SVM Hyperplane divides classes on either side 7
Figure 10: Radial SVM 8
Figure 11: Radial SVM transformation into kernels 8
Figure 12: General polynomial equation 8
Figure 13: Output boosted model equation 9
Figure 14: PCA explanation of variance of data set 10
Figure 15: Detailed variance of 42 predictors 11
Figure 16: Plot between coefficients & log-lambda for LASSO 11
Figure 17: Plot to calculate misclassification error rate 12
Figure 18: Variable importance plot of predictors in our data set 12
Figure 19: Variable importance plot using MARS model 13
Figure 20: Summary of resampled data using C.V. for parameter tuning using logistic regression 14
Figure 21: ROC for logistic regression 14
Figure 22: Confusion matrix 15
Figure 23: Plot between False and True positive rate 15
Figure 24:Summary of resampled data using C.V. for parameter tuning using KNN 15

Figure 25: Confusion matrix 16
Figure 26: Accuracy plot 16
Figure 27:Confusion matrix for LDA 16
Figure 28: LDA resampling summary 17
Figure 29:Summary of resampled data using C.V. for parameter tuning using SVM radial 18
Figure 30: Accuracy plot for repeated C.V. and resampled parameters 18
Figure 31: Confusion matrix for SVM linear technique with resampled parameters 19
Figure 32:Summary of resampled data using C.V. for parameter tuning using SVM Polynomial technique 20
Figure 33: Plot of accuracy using repeated cross-validation 21
Figure 34: Resampled parameters using bagging technique and confusion matrix 21
Figure 35: Confusion matrix 22
Figure 36: Accuracy used to select optimal model 22
Figure 37: Plot between false and true positive rate 23
Figure 38: Confusion matrix of parameters using boosting technique 23
Figure 39:Summary of resampled data using C.V. for parameter tuning using boosting technique 24
Figure 40: Confusion matrix for tuned and untuned parameter evaluation using neural networks 26
Figure 41: Confusion matrix for tuned and untuned parameter evaluation using Ensemble methods 26

Abstract

Cardiac arrhythmia is a condition of irregular heartbeat; beating either too slow or too fast. The heart rate which is too fast (above 100 bpm) is called tachycardia and the heart rate which is too slow (below 60 bpm) is called bradycardia, but in practice many types of arrhythmia have no symptoms which makes early diagnosis difficult. While most types of these diseases have no serious effects, some of them may cause stroke, heart failure or lead to other complications in a person's health if left untreated. It is therefore important to have a robust method or system which can diagnose arrhythmia in the early stages and save the lives of people. The objective of this paper is to classify patients' diagnosis into 16 classes, of which one class represents the absence of disease while the other 15 classes contain different subtypes of arrhythmia based on the patient's ECG records. The dataset being considered is taken from the University of California repository, main features of which have been selected using various methods. The data frame containing the significant predictors obtained from attribute selection was then subjected to training data to fit a predictive model and test data was used to compute the accuracy of the fitted model. It was found that the technique of SVM Radial yielded the optimal model post tuning, the misclassification error rate for which was found to be 17%.

Introduction

Arrhythmia can be either harmless or life threatening, and it therefore is very important to identify and analyse the onset of arrhythmia early to classify them successfully. It can be diagnosed by observing and analysing the output signals from ECG, which measures the function of the heart. The ECG setup consists of minimum four electrodes attached to different places such as chest or on body extremities such as arms or legs. The sensors in the ECG are wet and have a conductive gel which enhances the conductivity between skin and electrodes. Though the electricity generated is small, it is sufficient to be picked up by the electrodes and record the electrical functions of the heart.

ECG signals are predominantly made of P-waves, T-waves and QRS complex. These signals along with other predictor values of the patient such as age, weight, height and sex are used to predict the presence or absence of arrhythmia. However sometimes observing the signals minutely coupled with the other information of the patient can be tedious and there are chances of human error. In the emerging healthcare systems, development and establishment of a classification model that can correctly identify and classify a patient to different arrhythmia classes is thus important.

Here, we aim to use machine learning algorithm to automate the classification of different patients into 16 classes. Thus, the response column contains 16 classes with class 1, denoting the absence of arrhythmia while the other 15 classes denoting the presence of the disease. As a result, the response as class 1 was substituted as "No", while all the other responses were substituted as "Yes". The techniques implemented are bagging, random forest, ridge regression, LASSO, Linear & Quadratic Discriminant Analysis, KNN, Principal Component Analysis, Support Vector Machines and Polynomial Kernel.

Project Approach

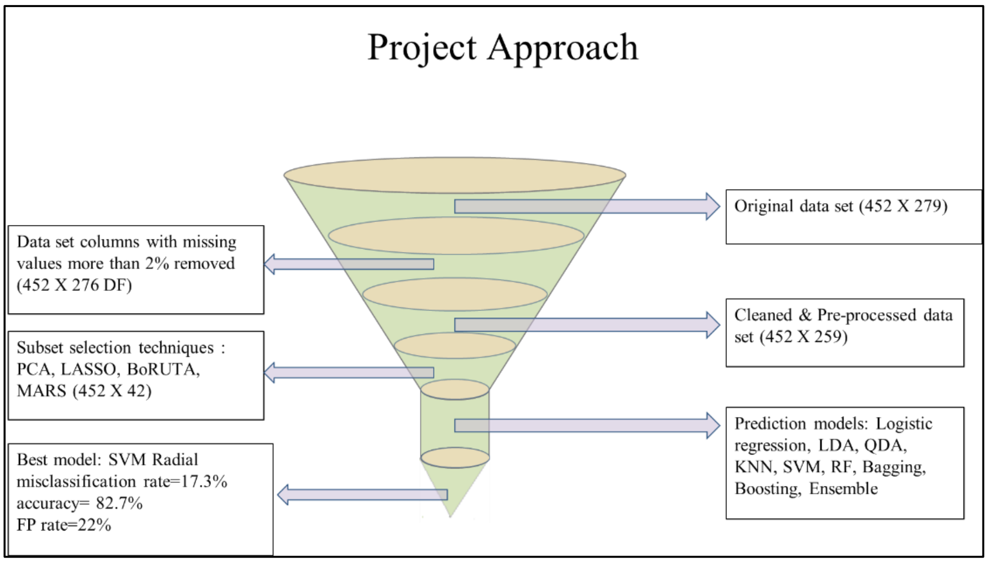

Project Approach
Original data set (452 X 279)
Data set columns with missing values more than 2% removed (452 X 276 DF)
Cleaned & Pre-processed data set (452 X 259)
Subset selection techniques : PCA, LASSO, BoRUTA, MARS (452 X 42)
Prediction models: Logistic regression, LDA, QDA, KNN, SVM, RF, Bagging, Boosting, Ensemble
Best model: SVM Radial misclassification rate=17.3% accuracy= 82.7% FP rate=22%

Figure 1: Project flow diagram

Data description

The data set is taken from the University of California, Irvine Machine learning data repository. There were initially 452 rows and 279 predictor values such as age, height, sex, weight and ECG characteristics which are important for classifying the patients. The data set is labelled in 16 classes. Class 1 signifies absence of the disease, i.e. has normal ECG; and the other 15 classes represent the subtypes of arrhythmia, i.e. symptoms of heart disease prevalent (discussed below). In this data set, class 1 has most of the data classified in it with 205 data responses and the other 247 ones are distributed among the other 15 types of arrhythmia. We also observed that there were no unclassified data. Class 1 refers to ECG normal, class 2-Ischemic changes, i.e. coronary heart disease prevalent, class 3-old anterior myocardial infarction, class 4-old inferior myocardial infarction, class 5-sinus tachycardia, class 6- sinus bradycardia, class 7-ventricular premature contraction (PVC), class 8-supraventricular premature contraction (SPC), class 9- left bundle branch block, class 10-right bundle branch block, class 11-first degree atrioventricular block and class 12- second degree atrioventricular block, class 13-third degree atrioventricular block, class 14-left ventricular hypertrophy, class 15-atrial fibrillation or flutter, class 16-rest unclassified. The main features have been identified and tabulated.

The challenge

Our fundamental aim is to detect the presence of different types of arrhythmia and classify them into either of the considered 16 classes. The number of observation or cases for establishing the classification model is limited as compared to the number of predictor values. It further has some missing values and some columns have same values, thus making the results obtained from the unprocessed data set more inclined/biased towards normal ECG.

Data Pre-processing

I. The dataset was analysed, and it was found that the complete dataset had a total of 408 missing values out of the entire data set of 452 X 279 values – comprising of 0.33% missing values.

II. Missing Values Correction: 3 predictors having more than 1% missing values in them were identified and removed (we restricted our threshold to 1% in our case), while 2 other missing values were substituted by column mean values. After removal of those columns, our data set was refined to 452 X 277 values.

III. Data set was then further cleaned by the following methods:
   a. Centring of data – The mean of all the observations was made to zero by subtracting each of the values from the corresponding column mean.
   b. Scaling of data – The data was scaled by adopting the standard deviation method where each observation was modified; adopting this method also did not affect the algorithm of subsequent methods.
   c. We then observed that 17 predictors had the same value for all observations and they were removed, thus our data set then transformed to a 452 X 260 valued one.

IV. The data set was then split into training and test data. We had to bifurcate the data in a way that a higher ratio of split would result in overfitting the data and a lower one would lead to underfitting them. Hence 70% data was training set and the rest 30% was test one, leading to a new data set with 316 X 42 values.

V. Conversion to Binary Class outcome: The response column contains 16 classes with class 1 - denoting absence of arrhythmia, while the other 15 classes denote the presence of disease. As a result, the response as class 1 was substituted as “No”, while all the other responses were substituted as “Yes”.

Implementation Details and Feature selection

Packages used

1. <u>Glmnet</u>: It is a package that fits a generalized linear model via penalized maximum likelihood. The regularization path is computed for the lasso or elastic net penalty at a grid of values for the regularization parameter lambda. It fits linear, logistic and multinomial, and Poisson models. A variety of predictions can be made from the fitted models. It can also fit multi-response linear regression. Glmnet solves the following problem

$$\min_{\beta_0,\beta} \frac{1}{N}\sum_{i=1}^{N} w_i l(y_i, \beta_0 + \beta^T x_i) + \lambda\left[(1-\alpha)||\beta||_2^2/2 + \alpha||\beta||_1\right],$$

Figure 2: Glmnet Equation

over a grid of values of λ covering the entire range. The tuning parameter λ controls the overall strength of the penalty. The package also makes use of the strong rules for efficient restriction of the active set. The package also includes methods for prediction and plotting, and a function that performs K-fold cross validation.

2. Caret: The caret package (short for Classification and REgression Training) contains functions to streamline the model training process for complex regression and classification problems. The package utilizes several R packages but tries not to load them all at package start-up (by removing formal package dependencies, the package start-up time can be greatly decreased).
3. Ggplot2: ggplot2 is a plotting system for R, based on the grammar of graphics, which tries to take the good parts of base and lattice graphics and none of the bad parts. It takes care of many of the fiddly details that make plotting a hassle (like drawing legends) as well as providing a powerful model of graphics that makes it easy to produce complex multi-layered graphics.
4. Random Forest: This package is used to implement Breiman's random forest algorithm (based on Breiman and Cutler's original Fortran code) for classification and regression. It can also be used in unsupervised mode for assessing proximities among data points.
5. Boruta: This package generally incorporates a wrapper algorithm which was leveraged for important feature selection in the dataset. This method basically incorporates a top-down search approach by comparing the importance of original attributes with the estimated importance parameters at random to eliminate the irrelevant features from the dataset.
6. BST: This package is generally used to support boosting for both classification and regression. This package authored by Zhu Wang is used to support gradient descent boosting for both hinge loss and square error loss. This package can be used for both convex and non-convex loss functions and was primarily used by us to facilitate tuning for boosting.
7. ADABAG: This method maintained by Esteban Alfaro encompasses multiclass AdaBoost, SAMME and Bagging. This algorithm uses classification trees as individual classifiers. After that, these classifiers can be used to predict new data. Using this, cross-validation estimation of the errors can be done and margin can also be varied.
8. fastADABOOST: This package in R generally is very fast and it generally uses decision trees as weak classifiers. This package only supports binary classification tasks. It supports both Adaboost.M1 algorithm and SAMME.R algorithm.
9. XGBOOST: This package generally stands for Extreme Gradient Boosting algorithm. It is a gradient Boosting framework which uses linear model and tree learning algorithm to make predictions. Its main advantage is that it is 10 times faster than classical gbm package and it can be used for parallel operations in both Windows and Linux with OpenMP.
10. IPRED: ipred package stands for Improved Predictors in R. It is used for improved predictive models for classification and regression problems as well as resampling based estimators of estimated error. It imports MASS, survival, nnet, class and prodlim for calculating these estimates. It is also used to estimate bootstrapped, control and cross-validated error estimates of the dataset.
11. pROC: This package in R is used to display and analyze ROC curves. This package in R contains all the tools for computing Receiver Operator Characteristics based on U-statistics or bootstrap. We can also find out the confidence intervals for these ROC curves using this package. The different parameters of the confusion matrix such as specificity and sensitivity can be computed from this package too.
12. ROCR: This package is used to visualize performance of many scoring classifiers. It is a flexible tool for creating 2D cut-off parameterized performance curves by combining any 2 from over 25 performance measures. ROC graphs, sensitivity, specificity, lift charts and precision plots which are the best measures of a classifier are contained in this package.

Feature Selection methods

1. PCA: This method is mostly used for unsupervised learning and hence removes the predicted values. 85% of total variance can be explained by 50-80 predictors in the data set. It is a method of feature extraction where the algorithm combines and transforms the input variables in such a way that the transformed variables are uncorrelated with each other. Here, we choose only those output variables which capture the highest variance of the data and drop the least important variables. The uncorrelated output variables are useful because the assumption of the linear model holds true as the linear model required the variables to be independent of each other. PCA makes the data easier to visualize and shows a strong suggestive pattern in the data set. It is also used when a reduction in number of variables is necessary but information about the important features that are to be eliminated is unknown. It can further be used when uncorrelated variables are to be generated and when output variables are less interpretable.
   For example, let us consider two-dimensional data of height and weight, now as we apply PCA to this data set, we get the combinations of the original variables in a new coordinate system in which every output variable has new x, y value. From these output variables, we choose those which capture a percentage variation of 80-85% of the data set.

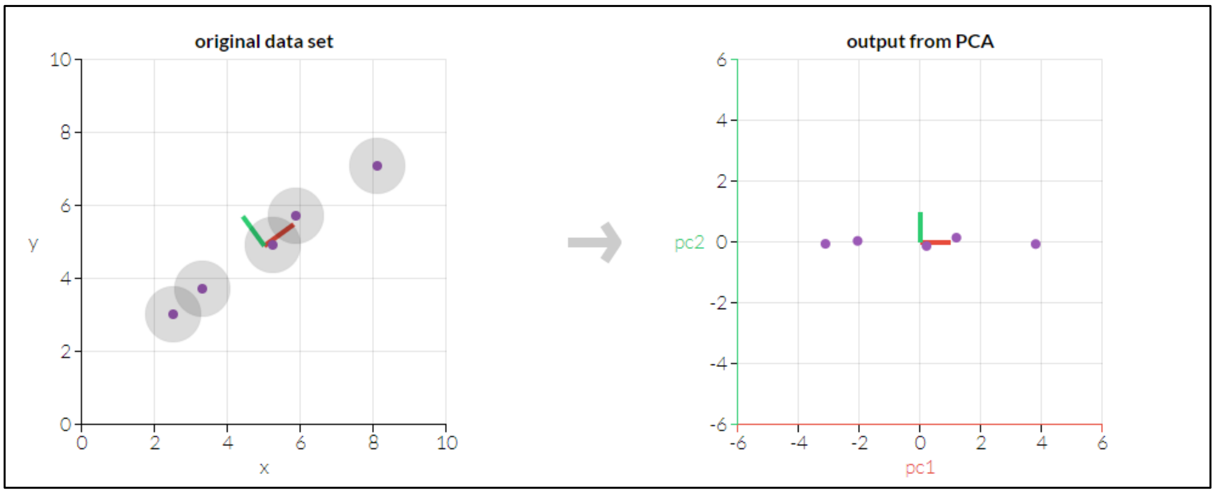


Figure 3: Before and After applying PCA technique to a sample data set

2. LASSO: The total number of significant predictors were identified to be 42 using this technique.
   The formulation of Lasso is as to minimise:

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = \text{RSS} + \lambda\sum_{j=1}^{p}|\beta_j|.$$

$$\underset{\beta}{\text{minimize}}\left\{\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2\right\} \quad \text{subject to} \quad \sum_{j=1}^{p}|\beta_j| \le s$$

Figure 4: LASSO equation minimization

In Lasso, some of the coefficients become exactly zero and thus we have fewer variables in the final model.

LASSO performs better when the response depends on fewer predictors which have significant importance and coefficients and others have small value or are zero.

3. Boruta: Algorithm works in a similar fashion as Lasso, the only difference being in addition to identifying the 42 significant predictors, it also identifies tentative predictors. In our case, the data set contained 19 tentative predictors.

4. Logistic Regression: It is a predictive analysis technique and a classification method that is used for analysing data sets in which there are independent input variables to determine the output/labels of the data. The dependent variable is this case is binary or dichotomous that is labelled as 1 (True, yes, healthy or success) or 0 (false, No, unhealthy or failure). The objective of logistic regression is to fit the best fitting model to the data set so that we can correctly find the relationship between the input and output/dependent variables to identify and classify the data to the output binary variable. The logistic regression transformation equation is like that of a linear regression, given as

$$\boxed{logit(p) = b_0 + b_1X_1 + b_2X_2 + b_3X_3 + \ldots + b_kX_k}$$

When P is the probability of presence of the label (P=1)

$$\boxed{odds = \frac{p}{1-p} = \frac{probability\ of\ presence\ of\ characteristic}{probability\ of\ absence\ of\ characteristic}}$$

Figure 5: Logistical regression transformation equation and corresponding probability

5. LDA: On applying LDA to our new data set, we observed an error "collinearity in predictors", i.e. a phenomenon in which one predictor variable in a regression model can be predicted (with a substantial degree of accuracy) linearly from other values. But the coefficient estimates may sometimes change erratically in response to certain small changes in the data set or model. Corrections were made to remove 3 collinearities in predictors and this technique was performed again.

   The assumptions made in this approach is that each variable is normally distributed and if the dataset has more than one predictor, then it is follows multivariate distribution.

   Suppose in each class, the predictor follows normal distribution in class k:

$$\boxed{k:\ X \sim N(\mu_k, \sigma_k^2)}$$

   and with equal variance, the density function of the predictor in class k is given by:

$$\boxed{f_k(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x-\mu_k)^2\right)}$$

Figure 6: Normal distribution and density function of predictor in a class

6. QDA: Quadratic Discriminant analysis works similar as LDA, however the assumption about the normality distribution and that the assumption that variance of each class is identical is relaxed. Also, the discriminant function takes a quadratic form.

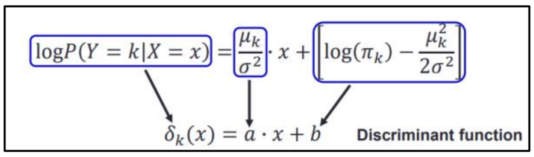

$\log P(Y = k|X = x) = \frac{\mu_k}{\sigma^2} \cdot x + \log(\pi_k) - \frac{\mu_k^2}{2\sigma^2}$
$\delta_k(x) = a \cdot x + b$
Discriminant function

Figure 7: QDA equation form

7. <u>KNN</u>

   If we are given a test observation X=x_0, KNN works according to the following steps:

   1. Identifying the k nearest neighbours, closest to x_0
   2. Calculating the class of those k neighbouring points
   3. Assigning x_o to the class label with majority fraction of the k neighbouring points class

   Suppose k=3, then as shown below for the point x, the 3 closest points are identified, among those 2 belong to blue class and 1 belongs to yellow class so, we assign x to the blue class

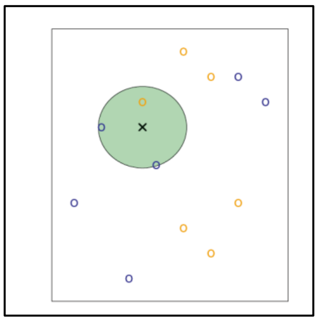


Figure 8: KNN depiction and class assignment

8. <u>Linear SVM</u>

   Support Vector Machine is a discriminative classifier which classifies the training data (supervised learning) into different categories by a separating hyperplane. The output of the algorithm gives the optimal hyperplane. If the dimension of the data set is p then the resulting hyperplane will be in p-1 dimension so, in two-dimensional space, the hyperplane will be a line which will divide the classes on its either side.

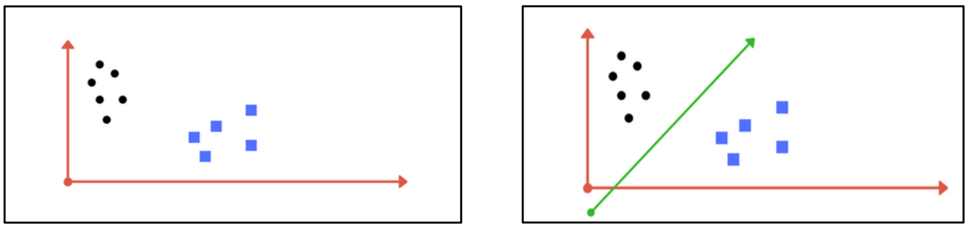


Figure 9: SVM Hyperplane divides classes on either side

9. <u>Radial SVM</u>: Here, we cannot draw any line to separate the classes and hence the need to apply the transformation, subsequently adding on one more dimension z-axis. Let's assume value of points on z plane, w = $x^2 + y^2$. If we plot the hyperplane in z axis, we can draw a clear separating line.

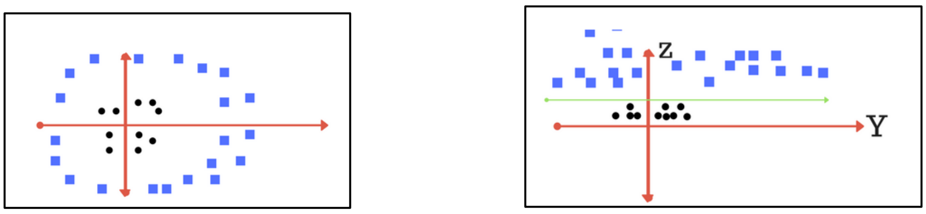


Figure 10: Radial SVM

When we transform back this line to original plane, it maps to circular boundary as shown in the image below. These transformations are called kernels.

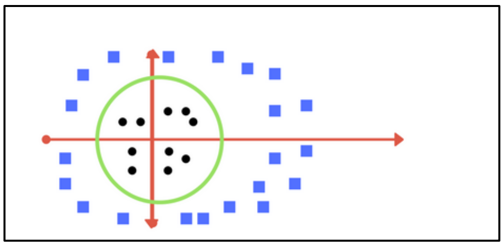


Figure 11: Radial SVM transformation into kernels

10. Polynomial SVM: SVM algorithm uses Kernels which is a mathematical function. It takes the input data and transforms it to the required form. There are different types of kernels for different algorithms.

    Equation is:

$$\boxed{k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d}$$

Figure 12: General polynomial equation

    Where d is the degree of the polynomial.

11. Bagging: It is abbreviated for bootstrap aggregating. It is a special case of model averaging and usually applied to decision tree methods. It also reduces variance and helps to avoid overfitting. If we are given a standard training set D of size n, bootstrap will generate m new training sets D_i, each of size s, by uniform sampling with replacement. Now, on these newly generated training sets, we apply regression/classification decision trees.

    Now suppose we want to predict the class for $X=x_0$, we will record the class label which each individual bootstrapped decision tree gives for $X=x_0$ and the observation is assigned the class label with majority vote.

12. Boosting: The working algorithm focuses on reducing bias and variance. Unlike bagging where a bootstrap approach to the sample grow the trees is applied, boosting rather modifies the tree grown on the original data set and thus, sequentially grows the decision tree. The boosting algorithm, therefore after each subsequent step, fits the residuals to improve f'. The output boosted model is

$$\boxed{\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)}$$

Figure 13: Output boosted model equation

13. Random Forest: It is a method of qualitative classification and regression tasks that labels the observation by constructing multiple decision trees from the training data set. The observation is then classified to the mode of the class prediction resulting from individual decision trees. It differs from bagging in a way that for each individual tree generated from the bootstrapped sample, each time a split in a tree is considered, a random sample of $\boldsymbol{m}$ predictors is chosen as split candidates from among the p predictors. (Usually $\boldsymbol{m} = \sqrt{\boldsymbol{p}}$ is used for classification)
14. Ensemble methods: It is a supervised algorithm which can be used on our data for prediction and classification models. Though there is a chance of over-fitting the data, ensemble methods like random decision trees and bagging have proven to yield better results on the dataset. Some common types of ensembles are Bayes Optimal Classifier, Bootstrap Aggregating, Boosting, Bayesian Parameter averaging, Bayesian model combination, Bucket of models and Stacking. Bucket of Models generally refers to cross-validation of the dataset and giving the optimal results for the best split. R, ScikitLearn in Python and MATLAB are used for implementing the ensemble methods.

15. Neural networks: Here, we have implemented a feed forward neural network which is different from recurrent neural networks. Here, the information moves in only one direction i.e. forward direction. They can be of many different types: single layer perceptron, multi-layer perceptron, Hopfield neural network, convolutional neural network, backpropagation network and feedback network. The parameters used in our case are decay where the values have been changed from 0.1 to 0.5 and the layers have been varied from 3 to 10 and the model predicting the best result was selected as the best model.

16. Tuning: It is a process which is used for finding the parameters for the best fit for our dataset. The tuning parameters vary according to the methods which we go on to use on the dataset like in case of ridge regression and lasso regression, lambda is the tuning parameter. The value of lambda determines the number of coefficients which we shrink to zero. When lambda is zero, it is the same as linear regression and when the value of lambda increases, the predictors shrink to zero. Here, in Caret R package, which we use for our project, we can tune all the parameters in a decision tree by using expand grid function and storing the parameters which we intend to tune while finding out the best fit.

Methodology:

1) Dealing with missing values in the data: The raw data frame was composed of 452 observations for a total of 279 predictor values that were recorded to detect the presence of Arrhythmia in a patient. The data was first cleaned by eliminating columns that contained more than 2% of missing values while for the remaining columns, the missing values were substituted by the respective column means. As a result, we generated a data frame of size 452X277 observations.
2) Pre-processing of data:
    a) Zero Variance Columns: The data was then pre-processed by eliminating columns with zero variance. These columns were removed because of their same value for all the observations which thereby eliminates their contribution towards classification. 17 such columns were found, and they were removed using the pre-process command found in the caret package.
    b) Scaling of Observations: The observations were also scaled by subtracting the column means and then dividing the result by column standard deviation for each of the values of the column. This was executed by using the pre-process command in the Caret Package.
    c) Conversion to Binary Class Outcomes: The given dataset had 16 output classes with output class 1 signifying absence of arrhythmia while the other 15 classes signifying the presence of it in various stages. Therefore, the output class 1 was converted to "No" while all the other remaining output classes were transformed to "Yes"
3) Splitting of data: The data was then split into training and testing data set. An optimal split of 70:30 was considered to save the data from overfitting as well as to ensure presence of decent number of observations that could constitute the training set.
4) Methods Used:
    a) *Principal Component Analysis*: The method of Principal Component Analysis is specifically suited for unsupervised learning. It was used to find the major principal components that can explain a certain proportion of variance for the data.

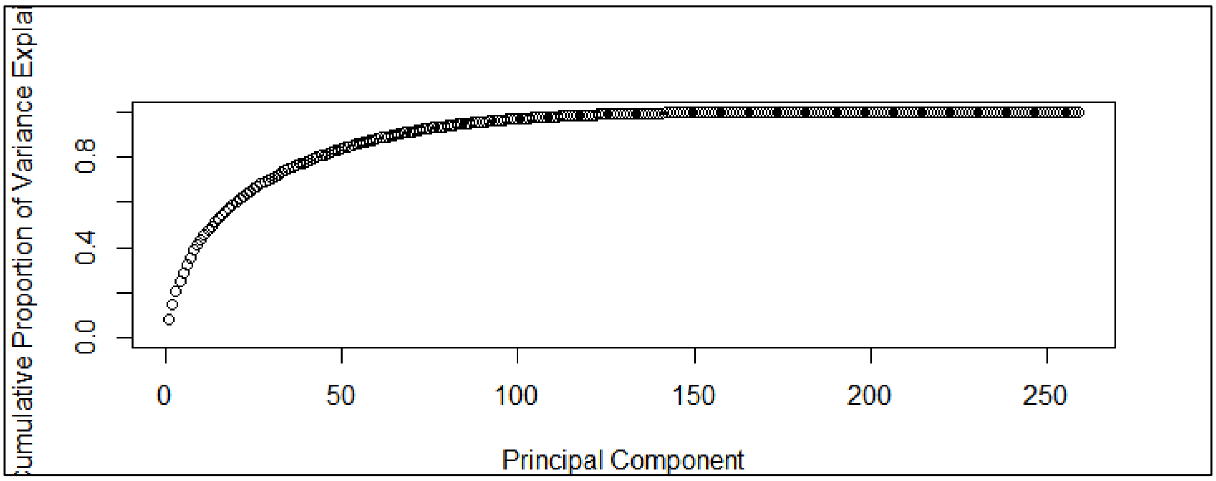
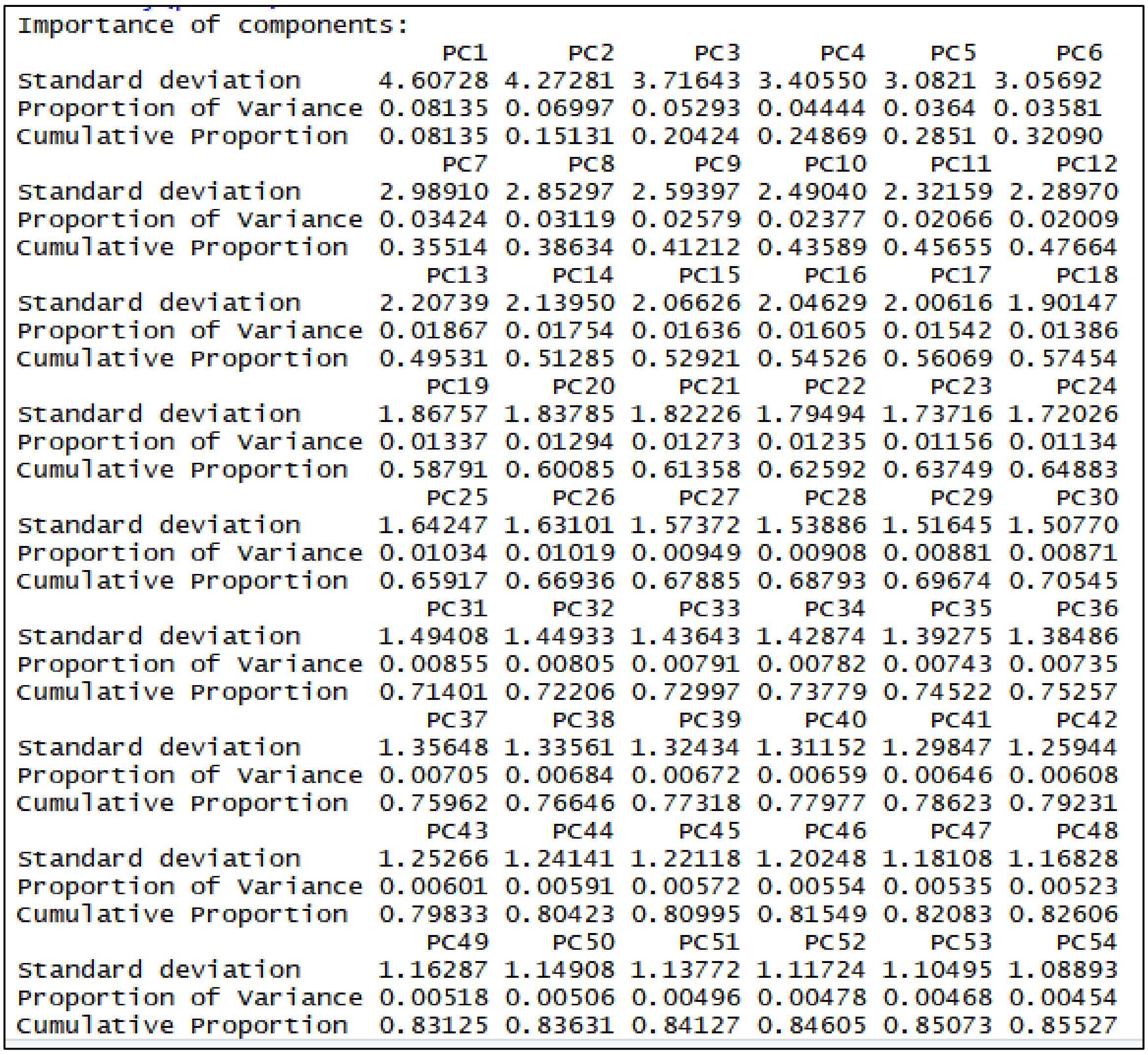


Figure 14: PCA explanation of variance of data set



```
Importance of components:
                          PC1     PC2     PC3     PC4     PC5     PC6
Standard deviation     4.60728 4.27281 3.71643 3.40550 3.0821 3.05692
Proportion of Variance 0.08135 0.06997 0.05293 0.04444 0.0364 0.03581
Cumulative Proportion  0.08135 0.15131 0.20424 0.24869 0.2851 0.32090
                          PC7     PC8     PC9    PC10    PC11    PC12
Standard deviation     2.98910 2.85297 2.59397 2.49040 2.32159 2.28970
Proportion of Variance 0.03424 0.03119 0.02579 0.02377 0.02066 0.02009
Cumulative Proportion  0.35514 0.38634 0.41212 0.43589 0.45655 0.47664
                         PC13    PC14    PC15    PC16    PC17    PC18
Standard deviation     2.20739 2.13950 2.06626 2.04629 2.00616 1.90147
Proportion of Variance 0.01867 0.01754 0.01636 0.01605 0.01542 0.01386
Cumulative Proportion  0.49531 0.51285 0.52921 0.54526 0.56069 0.57454
                         PC19    PC20    PC21    PC22    PC23    PC24
Standard deviation     1.86757 1.83785 1.82226 1.79494 1.73716 1.72026
Proportion of Variance 0.01337 0.01294 0.01273 0.01235 0.01156 0.01134
Cumulative Proportion  0.58791 0.60085 0.61358 0.62592 0.63749 0.64883
                         PC25    PC26    PC27    PC28    PC29    PC30
Standard deviation     1.64247 1.63101 1.57372 1.53886 1.51645 1.50770
Proportion of Variance 0.01034 0.01019 0.00949 0.00908 0.00881 0.00871
Cumulative Proportion  0.65917 0.66936 0.67885 0.68793 0.69674 0.70545
                         PC31    PC32    PC33    PC34    PC35    PC36
Standard deviation     1.49408 1.44933 1.43643 1.42874 1.39275 1.38486
Proportion of Variance 0.00855 0.00805 0.00791 0.00782 0.00743 0.00735
Cumulative Proportion  0.71401 0.72206 0.72997 0.73779 0.74522 0.75257
                         PC37    PC38    PC39    PC40    PC41    PC42
Standard deviation     1.35648 1.33561 1.32434 1.31152 1.29847 1.25944
Proportion of Variance 0.00705 0.00684 0.00672 0.00659 0.00646 0.00608
Cumulative Proportion  0.75962 0.76646 0.77318 0.77977 0.78623 0.79231
                         PC43    PC44    PC45    PC46    PC47    PC48
Standard deviation     1.25266 1.24141 1.22118 1.20248 1.18108 1.16828
Proportion of Variance 0.00601 0.00591 0.00572 0.00554 0.00535 0.00523
Cumulative Proportion  0.79833 0.80423 0.80995 0.81549 0.82083 0.82606
                         PC49    PC50    PC51    PC52    PC53    PC54
Standard deviation     1.16287 1.14908 1.13772 1.11724 1.10495 1.08893
Proportion of Variance 0.00518 0.00506 0.00496 0.00478 0.00468 0.00454
Cumulative Proportion  0.83125 0.83631 0.84127 0.84605 0.85073 0.85527
```

Figure 15: Detailed variance of 42 predictors

The graph shows a cumulative plot of the proportion of variance explained by all the attributes. The table above allows us to conclude that nearly 80% of the variance is explained using up to 42 predictors. (Detailed code has been attached in the Appendix).

b) *LASSO Technique*: It is preferred over Ridge Regression primarily because of its ability to support feature selection. The library Glmnet was used to execute the LASSO Command with the value of alpha being set to 1 and the corresponding family as "binomial".

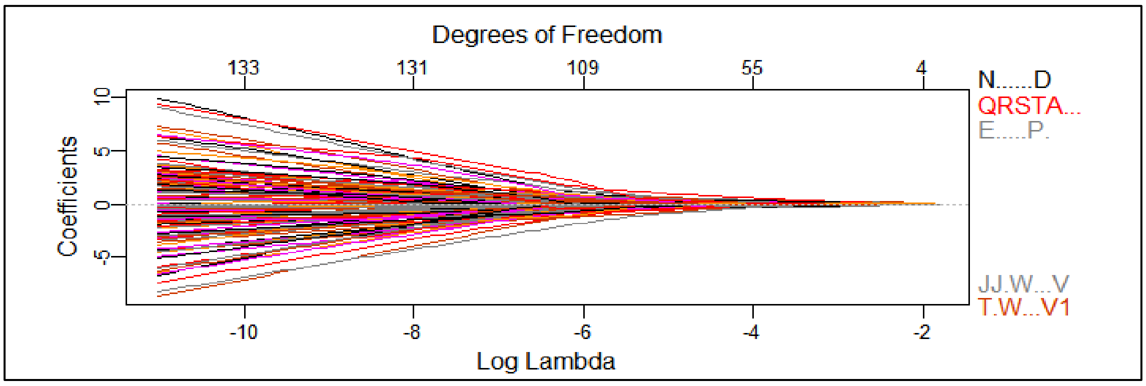


Figure 16: Plot between coefficients & log-lambda for LASSO

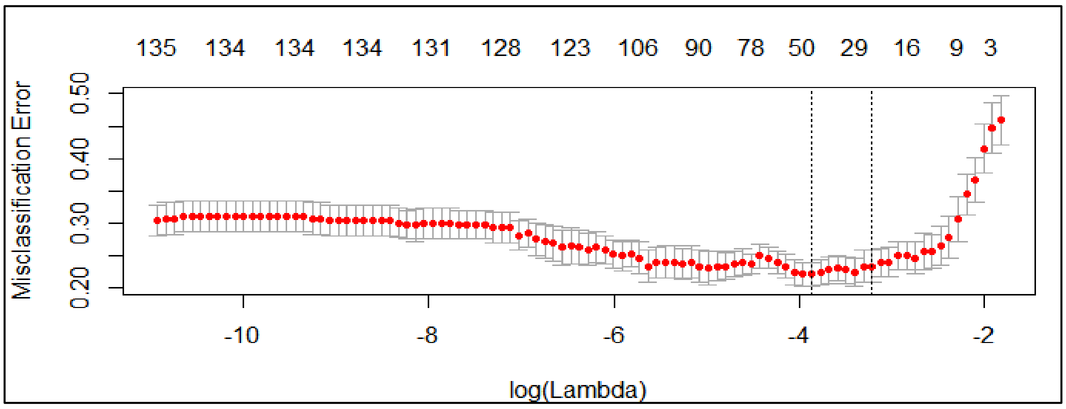


Figure 17: Plot to calculate misclassification error rate

The misclassification rate was plotted next as a function of log (lambda) to determine the key predictors having non-zero coefficient. The plots revealed that the minimum misclassification error was obtained corresponding to 42 predictors.

The following 42 predictors were identified as the major predictors: Sex, QRS.Duration, P.interval, Existence.of.diphasic.derivation.of.P.wave.of.channel.DI,Existence.of.diphasic.derivation.of.R.wave.of.channel.DII,Existence.of.ragged.P.wave.of.channel.DII,Existence.of.diphasic.derivation.of.T.wave.of.channel.DII,Existence.of.diphasic.derivation.of.P.wave.of.channel.DIII,Existence.of.diphasic.derivation.of.T.wave.of.channel.DIII,S.wave.of.channel.AVR,Existence.of.ragged.P.wave.of.channel.AVR,Number.of.intrinsic.deflections.of.channel.AVL,Q.wave.of.channel.AVF,S..wave.of.channel.AVF,Number.of.intrinsic.deflections.of.channel.AVF,S.wave.of.channel.V1,R..wave.of.channel.V1,Number.of.intrinsic.deflections.of.channel.V1,Existence.of.diphasic.derivation.of.R.wave.of.channel.V1,Q.wave.of.channel.V2,R..wave.of.channel.V2,Existence.of.diphasic.derivation.of.P.wave.of.channel.V2,Existence.of.diphasic.derivation.of.T.wave.of.channel.V2,Q.wave.of.channel.V3,Existence.of.ragged.R.wave.of.channel.V3,Existence.of.diphasic.derivation.of.T.wave.of.channel.V3, Q.wave.of.channel.V5, Number.of.intrinsic.deflections.of.channel.V5, S.wave.of.channel.V6, JJ.Wave.of.channel.DI,S.Wave.of.channel.DI,QRSTA.of.channel.DI,R.Wave.of.channel.DII,Q.Wave.of.channel.DIII,JJ.Wave.of.channel.AVR,QRSA.of.channel.AVR,P.Wave.of.channel.AVL,Q.Wave.of.channel.AVF,QRSTA.of.channel.V1,Q.Wave.of.channel.V2,JJ.Wave.of.channel.V5,T.Wave.of.channel.V6

c) <u>BORUTA Technique</u>: Boruta like LASSO is another widely used technique incorporated for the process of Feature Selection.

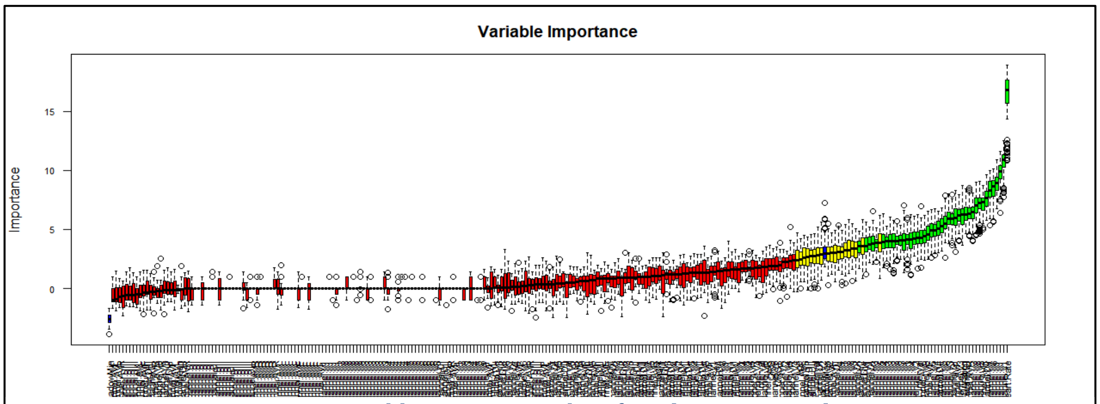


Figure 18: Variable importance plot of predictors in our data set

The Boruta technique also yielded 42 confirmed predictors and 19 tentative predictors which have been highlighted below:

Confirmed Important	Tentative important
QRS. Duration, T.Interval, Heart.Rate, Q.wave.of.channel.DII, Q.wave.of.chanel.AVF, S.wave.of.channel.V1, R.wave.of.channel.V1, Number.of.intrinsic.deflections.of.channel.V1,Q.wave.of.channel.V2, S.wave.of.channel.V2, R..wave.of.channel.V2, Number.of.intrinsic.deflections.of.channel.V2, R.wave.of.channel.V3, S.wave.of.channel.V3, Number.of.intrinsic.deflections.of.channel.V5, S.Wave.of.channel.DI, T.Wave.of.channel.DI, QRSA.of.channel.DI, QRSTA.of.channel.DI, Q.Wave.of.channel.DII, QRSTA.of.channel.DII, Q.Wave.of.channel.DIII, T.Wave.of.channel.AVR,QRSTA.of.channel.AVR, Q.Wave.of.channel.AVF, P.Wave.of.channel.V1, QRSTA.of.channel.V1, Q.Wave.of.channel.V2, S.Wave.of.channel.V2, R..Wave.of.channel.V2, T.Wave.of.channel.V2, QRSA.of.channel.V2, Q.Wave.of.channel.V3, S.Wave.of.channel.V3, QRSTA.of.channel.V3, JJ.Wave.of.channel.V4, R.Wave.of.channel.V4, S.Wave.of.channel.V4, QRSTA.of.channel.V4, JJ.Wave.of.channel.V5, T.Wave.of.channel.V5, T.Wave.of.channel.V6	Age, P.interval, S.wave.of.channel.DI, Number.of.intrinsic.deflections.of.channel.AVF, Q.wave.of.channel.V3, Number.of.intrinsic.deflections.of.channel.V3, JJ.Wave.of.channel.DI, T.Wave.of.channel.DII, JJ.Wave.of.channel.AVL, T.Wave.of.channel.AVL, R..Wave.of.channel.V1, QRSA.of.channel.V1, JJ.Wave.of.channel.V2, P.Wave.of.channel.V2, JJ.Wave.of.channel.V3, R.Wave.of.channel.V3, QRSA.of.channel.V3, JJ.Wave.of.channel.V6, QRSTA.of.channel.V6

d) MARS Model: MARS model was also used to check the feature selection. Required package called earth was pre-installed and it was used to create a variable importance plot.

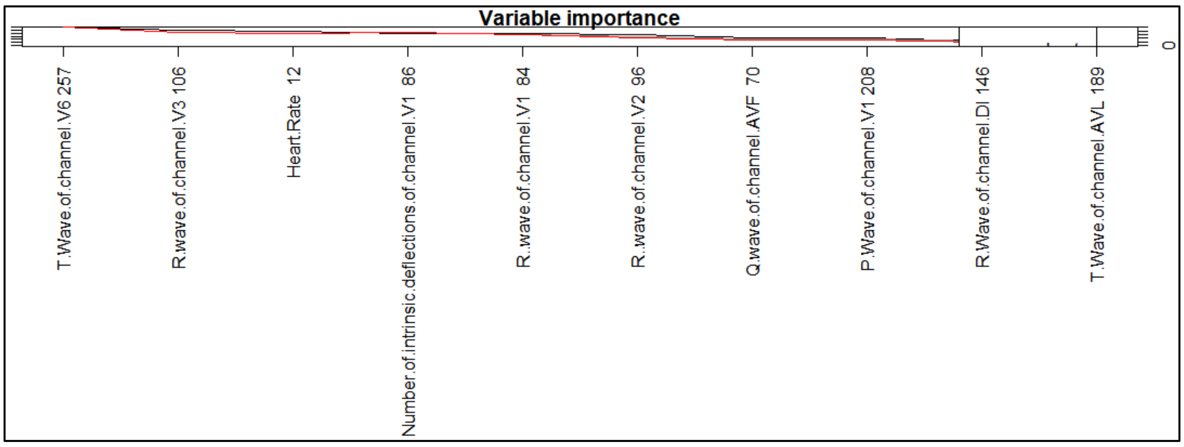


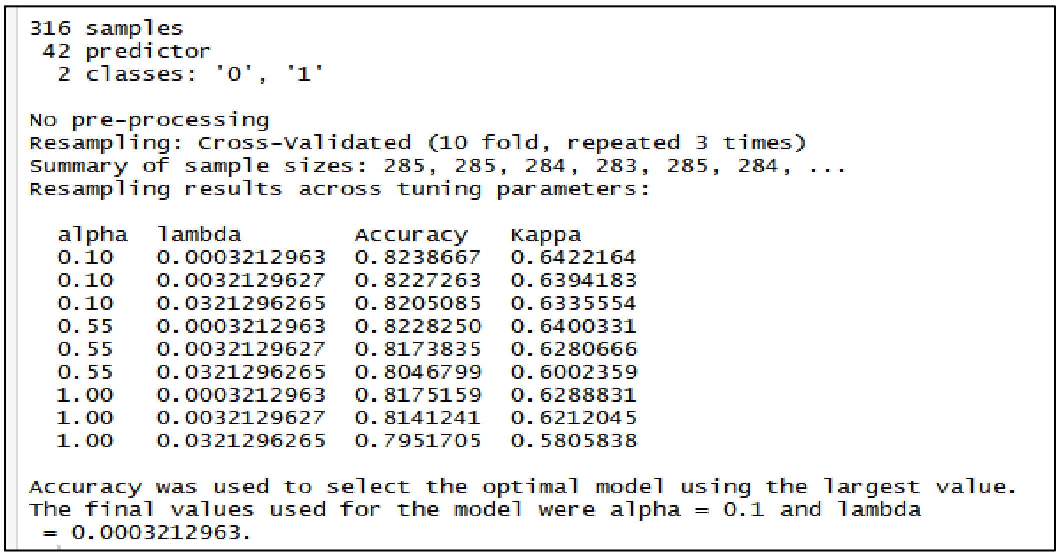
Figure 19: Variable importance plot using MARS model

5) Dataset with feature selection: Based on the usage of above techniques, it was found that the method of LASSO was able to yield the smallest misclassification error corresponding to 42 attributes. As a result, these 42 predictors along with the class attribute were used to generate a new data frame of 452 observations against these 42 predictors and one response value.

6) Prediction Models: The prediction models were created for different methods by using the Caret Package. The function "TrainControl" governs how models are created with a sub function of method deciding the resampling method while the number and repeats controlling the number of K-folds in Cross validation or number of resamplings for the bootstrapping technique. Another sub-function in the form of metric is also provided which determines the performance metric based on which the optimal model is chosen.

- Logistic Regression

The training data set composed of 316 observations for 42 predictors and 1 response were used to train the model by using the method glmnet which tunes the model for alpha and lambda.



```
316 samples
 42 predictor
  2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 285, 285, 284, 283, 285, 284, ...
Resampling results across tuning parameters:

  alpha  lambda        Accuracy   Kappa
  0.10   0.0003212963  0.8238667  0.6422164
  0.10   0.0032129627  0.8227263  0.6394183
  0.10   0.0321296265  0.8205085  0.6335554
  0.55   0.0003212963  0.8228250  0.6400331
  0.55   0.0032129627  0.8173835  0.6280666
  0.55   0.0321296265  0.8046799  0.6002359
  1.00   0.0003212963  0.8175159  0.6288831
  1.00   0.0032129627  0.8141241  0.6212045
  1.00   0.0321296265  0.7951705  0.5805838

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were alpha = 0.1 and lambda
 = 0.0003212963.
```

Figure 20: Summary of resampled data using C.V. for parameter tuning using logistic regression

The ROC Curve was plotted with False Positive Rate on the x-axis and True Positive Rate on the y-axis. The prediction method was tuned, and it was observed that the best model based on accuracy corresponded to a threshold of 0.381 with a prediction accuracy of 76.47% (which means a misclassification error rate of 23.53%). The area under the curve as a metric was also computed and it was found to be equal to 0.705. The False Positive Rate being our key performance indicator was found to be equal to 30.77%.

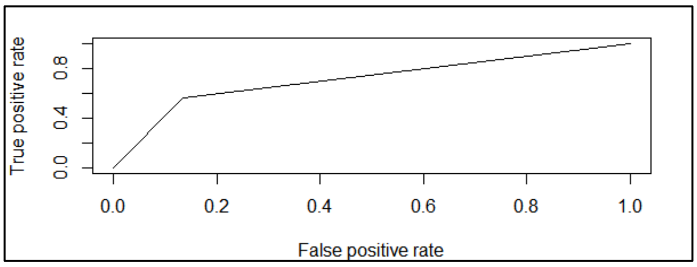


Figure 21: ROC for logistic regression

- KNN
  The training data set composed of 316 observations for 42 predictors and 1 response were used to train the model by using the method KNN which tunes the model for the optimal value of k.

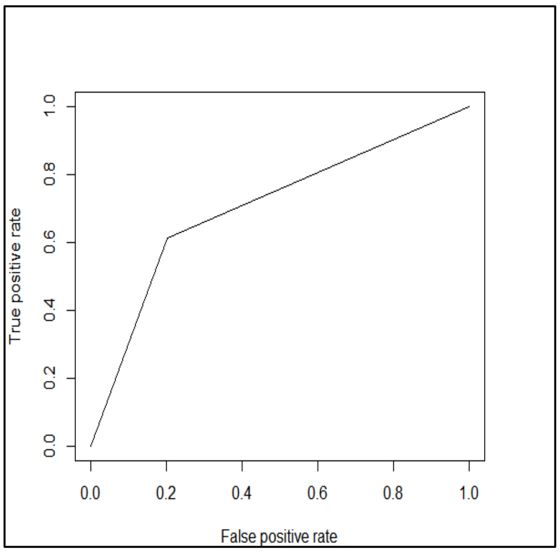
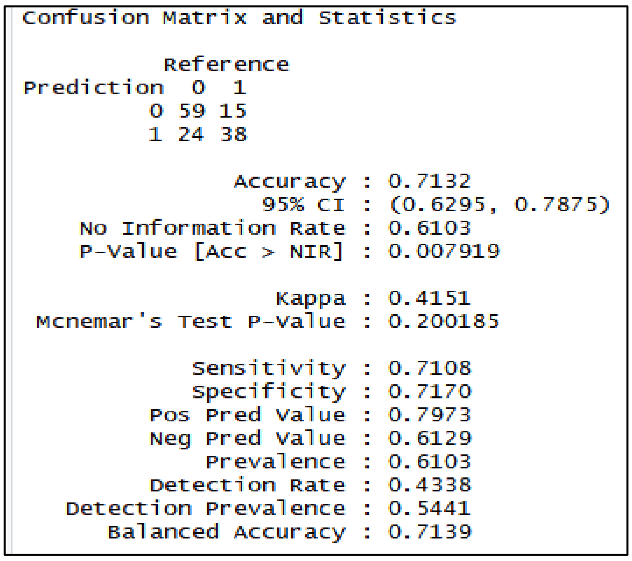
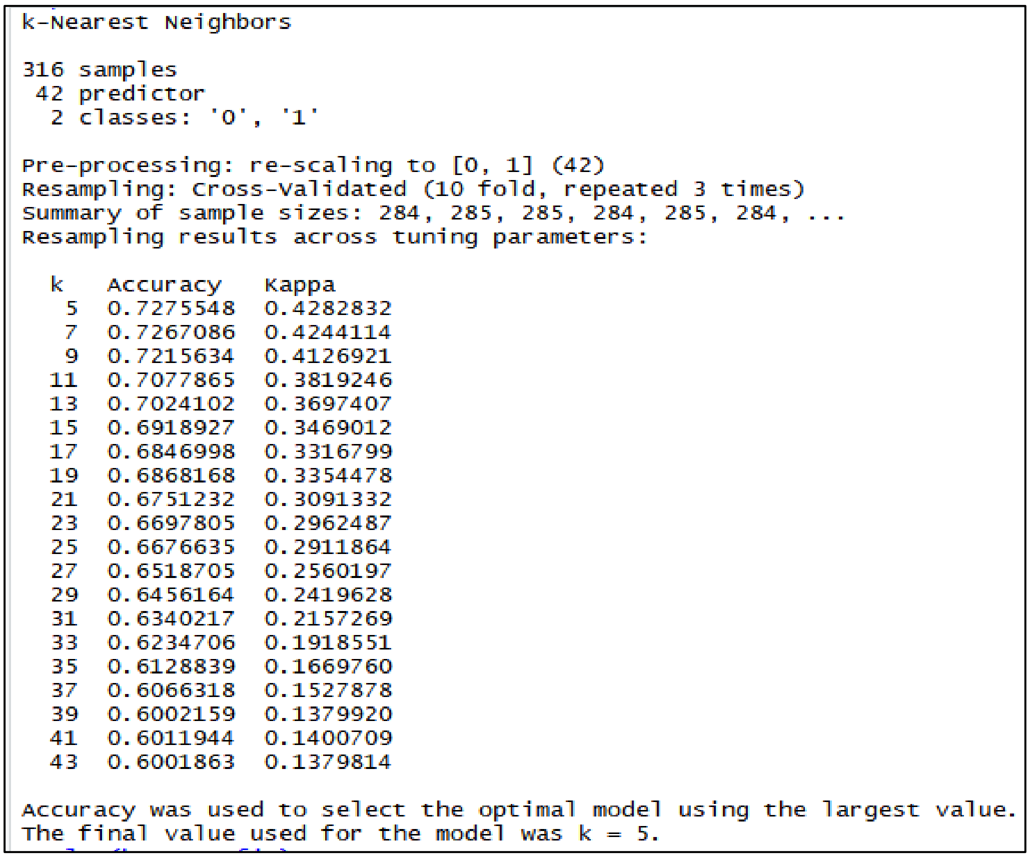


Figure 23: Plot between False and True positive rate



```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 59 15
         1 24 38

               Accuracy : 0.7132
                 95% CI : (0.6295, 0.7875)
    No Information Rate : 0.6103
    P-Value [Acc > NIR] : 0.007919

                  Kappa : 0.4151
 Mcnemar's Test P-Value : 0.200185

            Sensitivity : 0.7108
            Specificity : 0.7170
         Pos Pred Value : 0.7973
         Neg Pred Value : 0.6129
             Prevalence : 0.6103
         Detection Rate : 0.4338
   Detection Prevalence : 0.5441
      Balanced Accuracy : 0.7139
```

Figure 22: Confusion matrix



```
k-Nearest Neighbors

316 samples
 42 predictor
  2 classes: '0', '1'

Pre-processing: re-scaling to [0, 1] (42)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 284, 285, 285, 284, 285, 284, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   5  0.7275548  0.4282832
   7  0.7267086  0.4244114
   9  0.7215634  0.4126921
  11  0.7077865  0.3819246
  13  0.7024102  0.3697407
  15  0.6918927  0.3469012
  17  0.6846998  0.3316799
  19  0.6868168  0.3354478
  21  0.6751232  0.3091332
  23  0.6697805  0.2962487
  25  0.6676635  0.2911864
  27  0.6518705  0.2560197
  29  0.6456164  0.2419628
  31  0.6340217  0.2157269
  33  0.6234706  0.1918551
  35  0.6128839  0.1669760
  37  0.6066318  0.1527878
  39  0.6002159  0.1379920
  41  0.6011944  0.1400709
  43  0.6001863  0.1379814

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 5.
```

Figure 24:Summary of resampled data using C.V. for parameter tuning using KNN

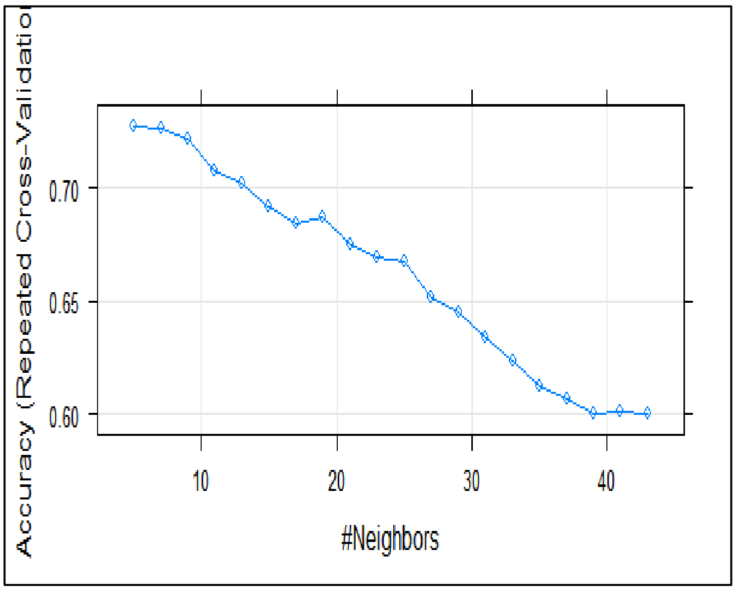
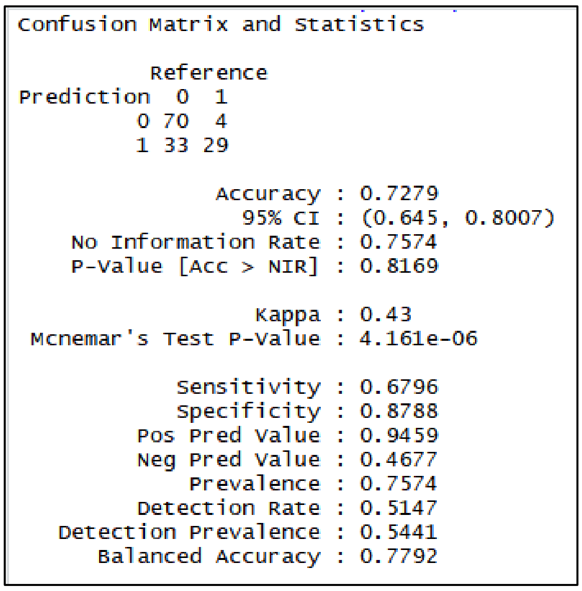


Figure 26: Accuracy plot

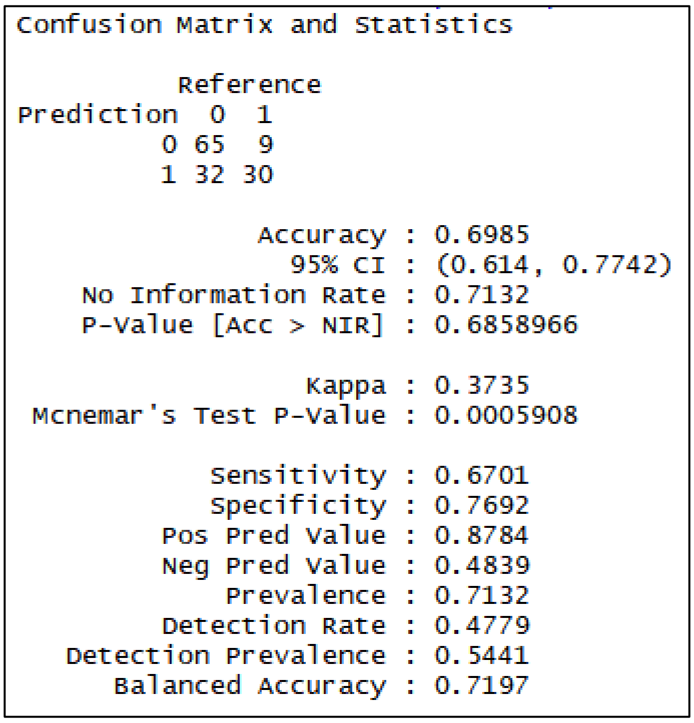


```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 70  4
         1 33 29

               Accuracy : 0.7279
                 95% CI : (0.645, 0.8007)
    No Information Rate : 0.7574
    P-Value [Acc > NIR] : 0.8169

                  Kappa : 0.43
 Mcnemar's Test P-Value : 4.161e-06

            Sensitivity : 0.6796
            Specificity : 0.8788
         Pos Pred Value : 0.9459
         Neg Pred Value : 0.4677
             Prevalence : 0.7574
         Detection Rate : 0.5147
   Detection Prevalence : 0.5441
      Balanced Accuracy : 0.7792
```

Figure 25: Confusion matrix

ROC Curve was plotted with False Positive Rate on the x-axis and True Positive Rate on the y-axis. The prediction method was tuned, and it was observed that the best model based on accuracy obtained via repeated cross-validation corresponded to K=5, i.e. 5-nearest neighbours with a prediction accuracy of 72.79% (which means a misclassification error rate of 27.21%). The False Positive Rate being our key performance indicator was found to be equal to 12.12%.

- LDA
  The training data set composed of 316 observations for 42 predictors and 1 response were used to train the model by using the method LDA which tunes the model for dimen as the tuning parameter.



```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 65  9
         1 32 30

               Accuracy : 0.6985
                 95% CI : (0.614, 0.7742)
    No Information Rate : 0.7132
    P-Value [Acc > NIR] : 0.6858966

                  Kappa : 0.3735
 Mcnemar's Test P-Value : 0.0005908

            Sensitivity : 0.6701
            Specificity : 0.7692
         Pos Pred Value : 0.8784
         Neg Pred Value : 0.4839
             Prevalence : 0.7132
         Detection Rate : 0.4779
   Detection Prevalence : 0.5441
      Balanced Accuracy : 0.7197
```

Figure 27:Confusion matrix for LDA

The confusion matrix yields a relatively low value of accuracy of 69.85% corresponding to the tuned model. It was observed however in the warnings section that there is a high degree of correlation present among a few predictors responsible for the relatively high misclassification error rate.

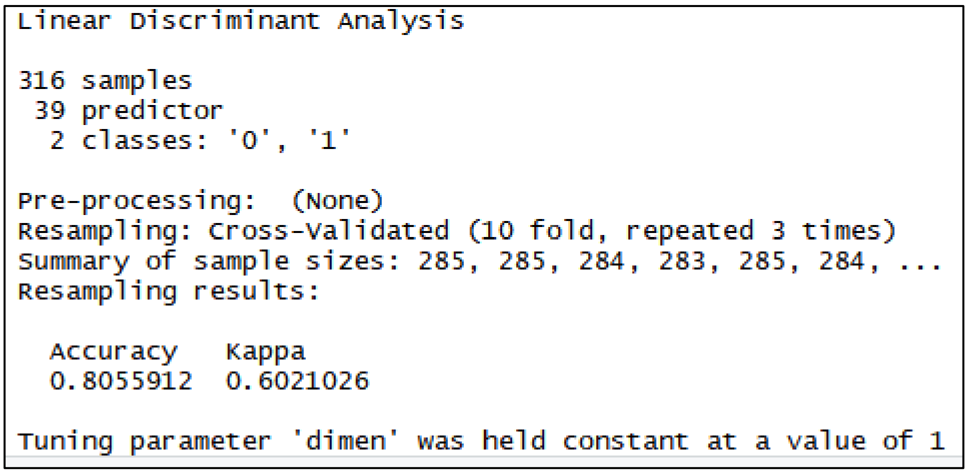
Therefore, the predictors data set was further revised to remove the highly collinear predictors by setting a threshold value of 0.7 which means that if the degree of collinearity between two predictors was found to be greater than or equal to 0.7, then those predictors were removed from further analysis.

Features found to be containing high degree of correlation were:

1) "Existence.of.diphasic.derivation.of.R.wave.of.channel.V1"

2) "Number.of.intrinsic.deflections.of.channel.AVL"

3) "Existence.of.diphasic.derivation.of.R.wave.of.channel.DII"

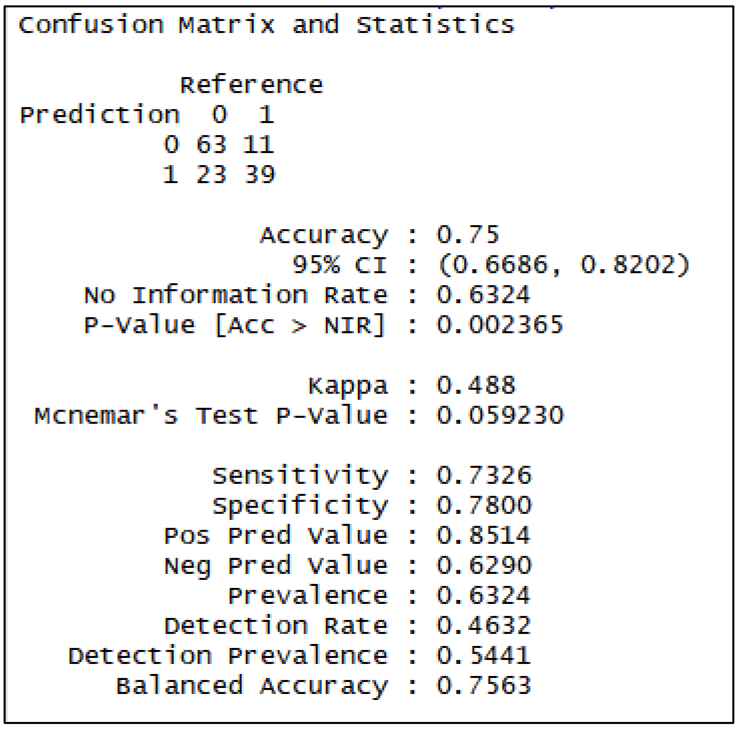


```
Linear Discriminant Analysis

316 samples
 39 predictor
  2 classes: '0', '1'

Pre-processing:  (None)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 285, 285, 284, 283, 285, 284, ...
Resampling results:

  Accuracy   Kappa
  0.8055912  0.6021026

Tuning parameter 'dimen' was held constant at a value of 1
```

Figure 28: LDA resampling summary

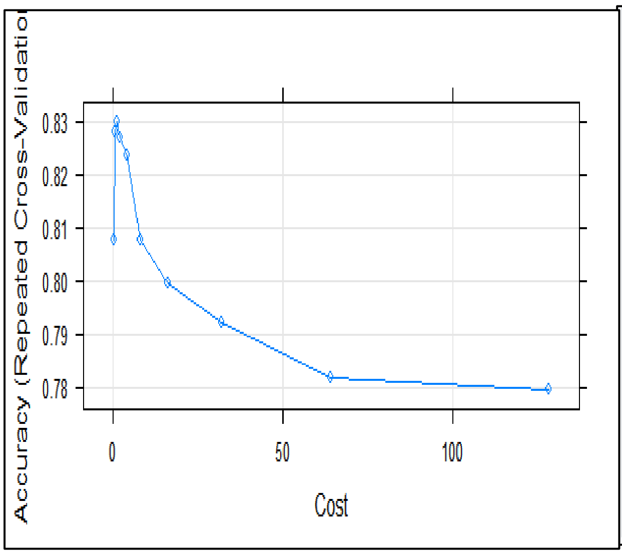
The model's accuracy was found to increase significantly to 80.56% upon removing the correlated attributes while the value of specificity (our key performance indicator) was found to become 23.08%

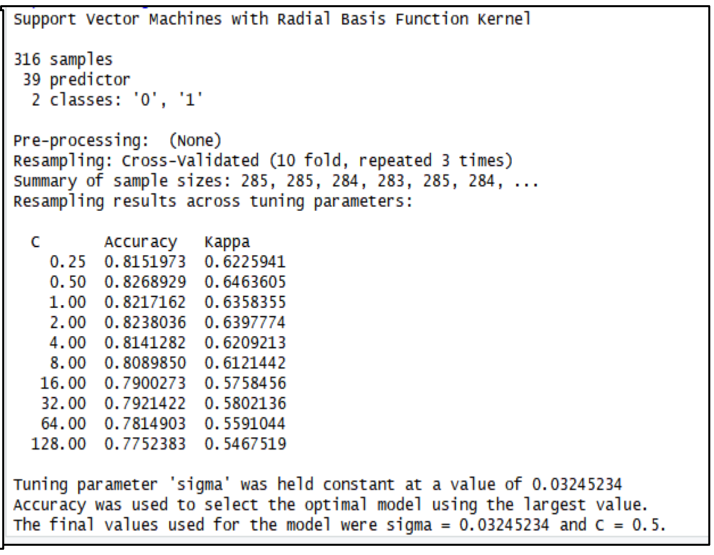
- SVM Radial
  The training data set composed of 316 observations for 42 predictors and 1 response were used to train the model by using the method SVMRadial which tunes the model for cost and sigma as the tuning parameters.



```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 63 11
         1 23 39

               Accuracy : 0.75
                 95% CI : (0.6686, 0.8202)
    No Information Rate : 0.6324
    P-Value [Acc > NIR] : 0.002365

                  Kappa : 0.488
 Mcnemar's Test P-Value : 0.059230

            Sensitivity : 0.7326
            Specificity : 0.7800
         Pos Pred Value : 0.8514
         Neg Pred Value : 0.6290
             Prevalence : 0.6324
         Detection Rate : 0.4632
   Detection Prevalence : 0.5441
      Balanced Accuracy : 0.7563
```

Figure 29:Summary of resampled data using C.V. for parameter tuning using SVM radial



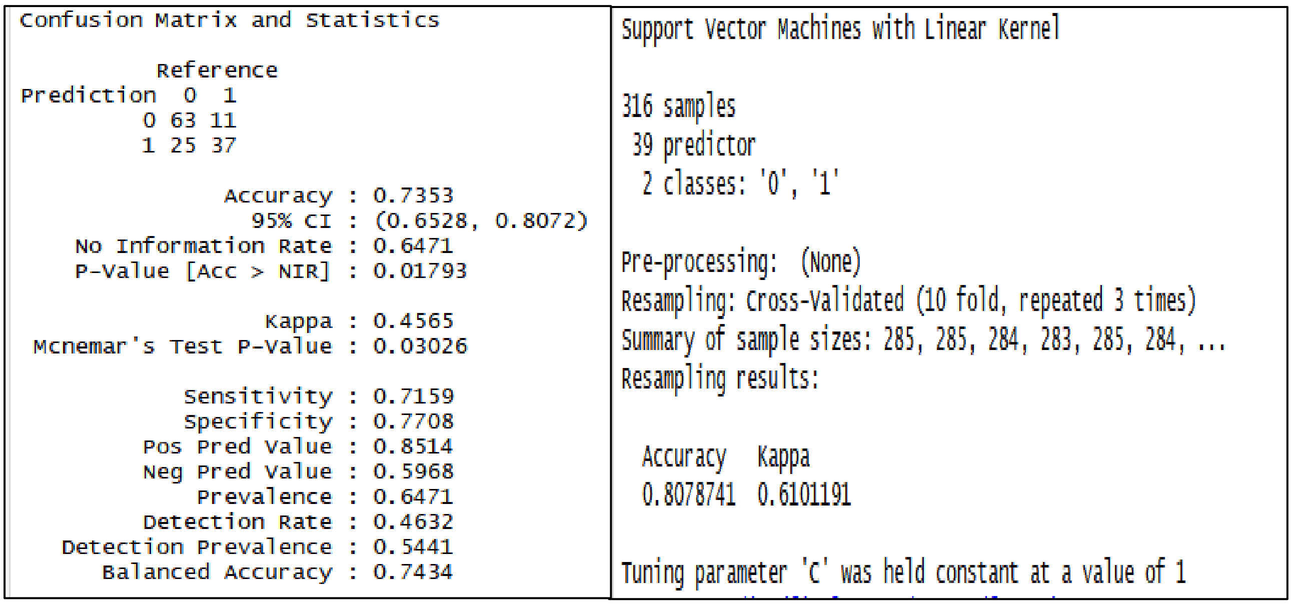


```
Support Vector Machines with Radial Basis Function Kernel

316 samples
 39 predictor
  2 classes: '0', '1'

Pre-processing:  (None)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 285, 285, 284, 283, 285, 284, ...
Resampling results across tuning parameters:

  C       Accuracy   Kappa
    0.25  0.8151973  0.6225941
    0.50  0.8268929  0.6463605
    1.00  0.8217162  0.6358355
    2.00  0.8238036  0.6397774
    4.00  0.8141282  0.6209213
    8.00  0.8089850  0.6121442
   16.00  0.7900273  0.5758456
   32.00  0.7921422  0.5802136
   64.00  0.7814903  0.5591044
  128.00  0.7752383  0.5467519

Tuning parameter 'sigma' was held constant at a value of 0.03245234
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.03245234 and C = 0.5.
```

Figure 30: Accuracy plot for repeated C.V. and resampled parameters

The accuracy was found to be 75%, which corresponds to a misclassification error rate of 25%. However, upon tuning the model through a grid function composed of multiple cost functions as shown above, the maximum accuracy achieved was equal to 82.68 % corresponding to a cost value of 0.5 and sigma value of 0.03. The False Positive Rate being our key performance indicator was found to be equal to 22%.

- <u>SVM Linear</u>
  The training data set composed of 316 observations for 42 predictors and 1 response were used to train the model by using the method SVMLinear which tunes the model for cost as the tuning parameters.



```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 63 11
         1 25 37

               Accuracy : 0.7353
                 95% CI : (0.6528, 0.8072)
    No Information Rate : 0.6471
    P-Value [Acc > NIR] : 0.01793

                  Kappa : 0.4565
 Mcnemar's Test P-Value : 0.03026

            Sensitivity : 0.7159
            Specificity : 0.7708
         Pos Pred Value : 0.8514
         Neg Pred Value : 0.5968
             Prevalence : 0.6471
         Detection Rate : 0.4632
   Detection Prevalence : 0.5441
      Balanced Accuracy : 0.7434
```

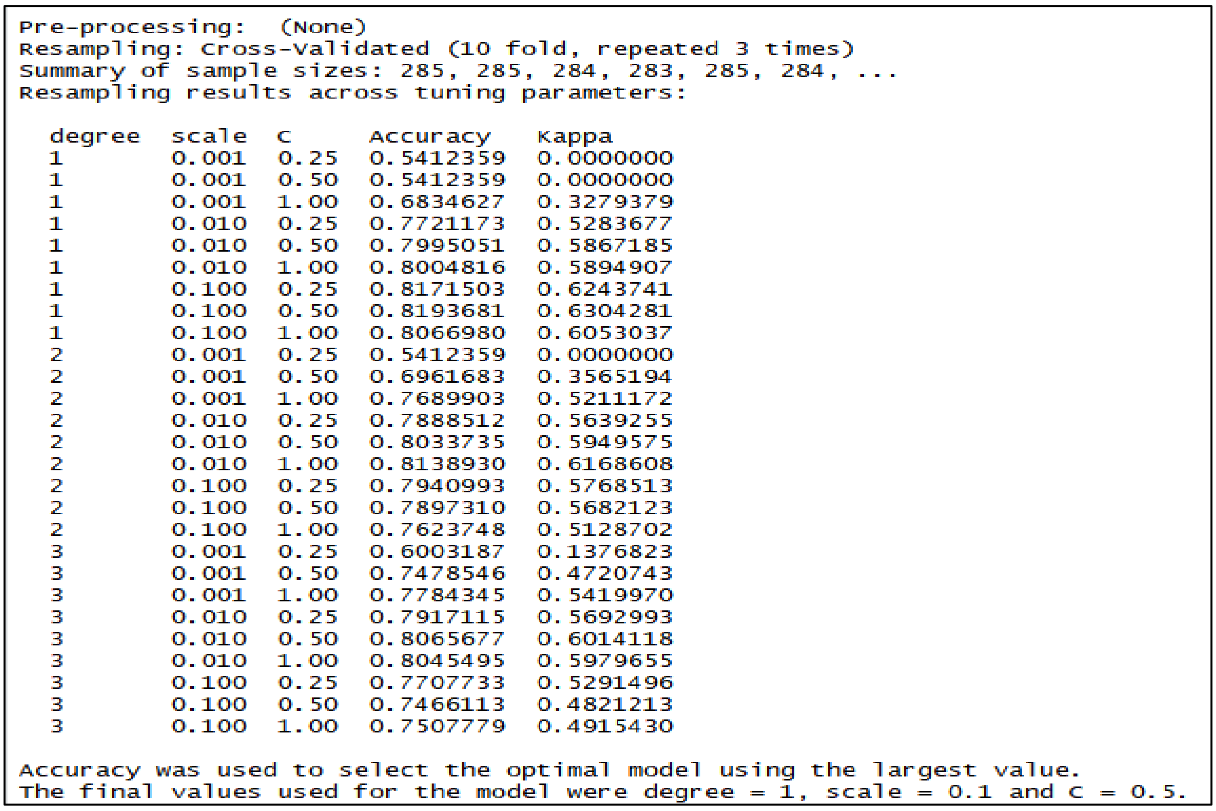
```
Support Vector Machines with Linear Kernel

316 samples
 39 predictor
  2 classes: '0', '1'

Pre-processing:  (None)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 285, 285, 284, 283, 285, 284, ...
Resampling results:

  Accuracy   Kappa
  0.8078741  0.6101191

Tuning parameter 'C' was held constant at a value of 1
```

Figure 31: Confusion matrix for SVM linear technique with resampled parameters

It was observed that the model yielded an accuracy of 73.53% which upon tuning increased to 80.78%. The value of False Positive Rate which was our key performance indicator was also found to be equal to 22.92%

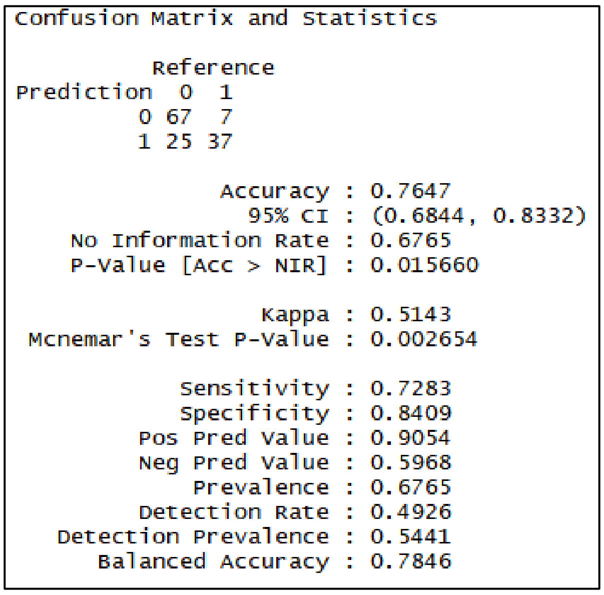
- SVM Polynomial

The training data set composed of 316 observations for 42 predictors and 1 response were used to train the model by using the method SVMPoly which tunes the model for cost, degree and scale as the tuning parameters.



```
Pre-processing:  (None)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 285, 285, 284, 283, 285, 284, ...
Resampling results across tuning parameters:

  degree  scale  C     Accuracy   Kappa
  1       0.001  0.25  0.5412359  0.0000000
  1       0.001  0.50  0.5412359  0.0000000
  1       0.001  1.00  0.6834627  0.3279379
  1       0.010  0.25  0.7721173  0.5283677
  1       0.010  0.50  0.7995051  0.5867185
  1       0.010  1.00  0.8004816  0.5894907
  1       0.100  0.25  0.8171503  0.6243741
  1       0.100  0.50  0.8193681  0.6304281
  1       0.100  1.00  0.8066980  0.6053037
  2       0.001  0.25  0.5412359  0.0000000
  2       0.001  0.50  0.6961683  0.3565194
  2       0.001  1.00  0.7689903  0.5211172
  2       0.010  0.25  0.7888512  0.5639255
  2       0.010  0.50  0.8033735  0.5949575
  2       0.010  1.00  0.8138930  0.6168608
  2       0.100  0.25  0.7940993  0.5768513
  2       0.100  0.50  0.7897310  0.5682123
  2       0.100  1.00  0.7623748  0.5128702
  3       0.001  0.25  0.6003187  0.1376823
  3       0.001  0.50  0.7478546  0.4720743
  3       0.001  1.00  0.7784345  0.5419970
  3       0.010  0.25  0.7917115  0.5692993
  3       0.010  0.50  0.8065677  0.6014118
  3       0.010  1.00  0.8045495  0.5979655
  3       0.100  0.25  0.7707733  0.5291496
  3       0.100  0.50  0.7466113  0.4821213
  3       0.100  1.00  0.7507779  0.4915430

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were degree = 1, scale = 0.1 and C = 0.5.
```

Figure 32:Summary of resampled data using C.V. for parameter tuning using SVM Polynomial technique



```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 67  7
         1 25 37

               Accuracy : 0.7647
                 95% CI : (0.6844, 0.8332)
    No Information Rate : 0.6765
    P-Value [Acc > NIR] : 0.015660

                  Kappa : 0.5143
 Mcnemar's Test P-Value : 0.002654

            Sensitivity : 0.7283
            Specificity : 0.8409
         Pos Pred Value : 0.9054
         Neg Pred Value : 0.5968
             Prevalence : 0.6765
         Detection Rate : 0.4926
   Detection Prevalence : 0.5441
      Balanced Accuracy : 0.7846
```

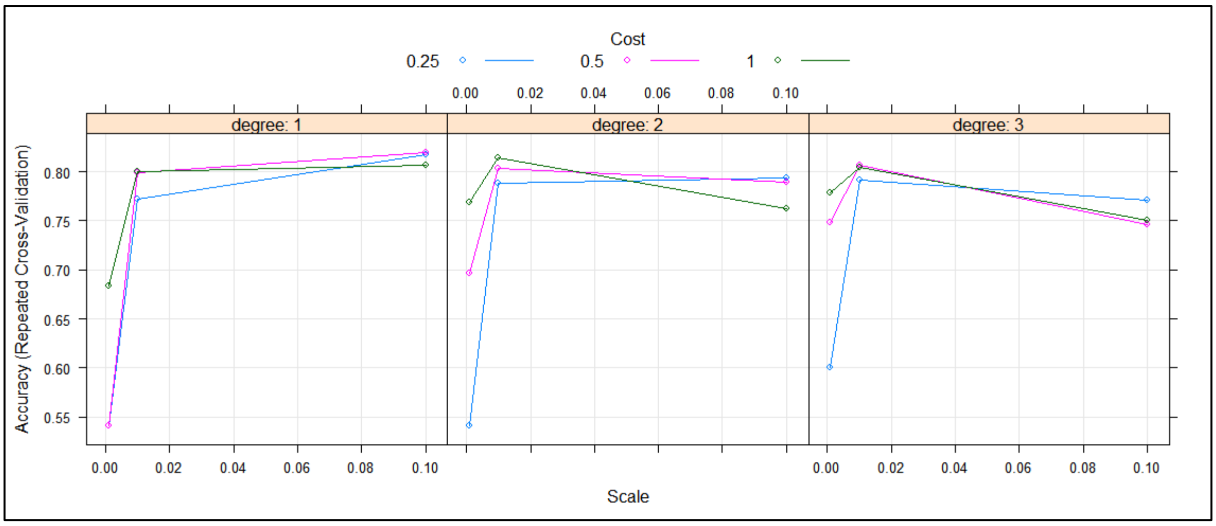
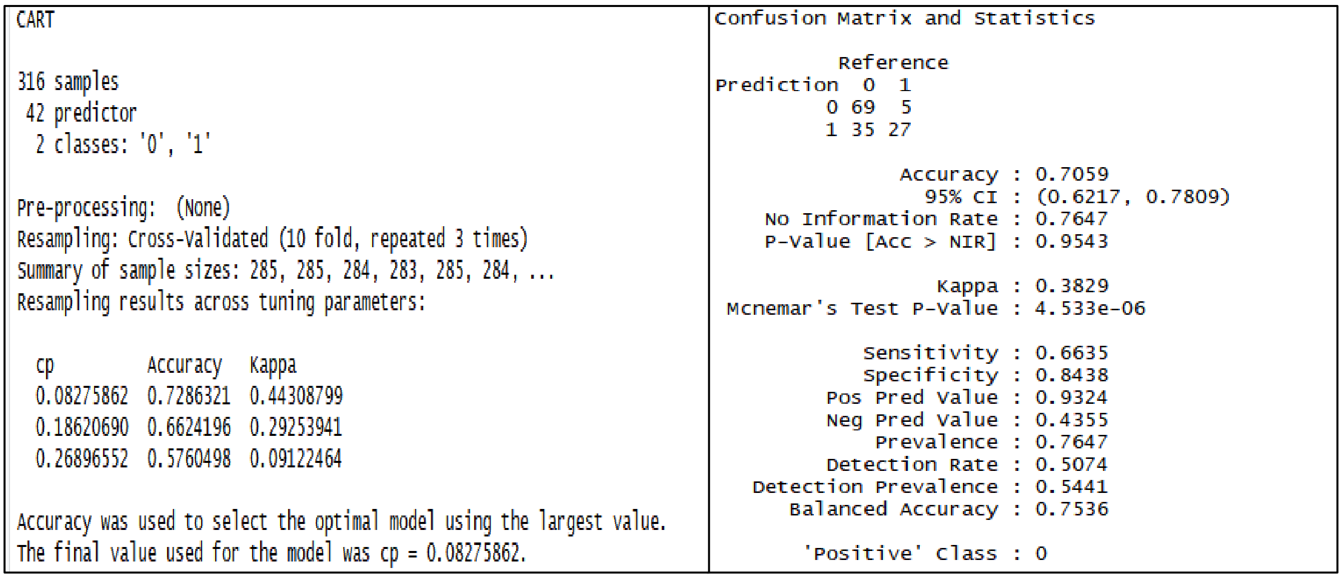


Figure 33: Plot of accuracy using repeated cross-validation

It was observed that the model depicted an accuracy of 76.47% when not tuned for any of the hyperparameters. However, as soon as we were able to tune the model for the value of cost, degree and scale, the best model based on accuracy was obtained which had 81.93% (or misclassification error rate of 18.07%) corresponding to degree =1, scale = 0.1 and cost =0.5

- <u>Bagging</u>
  The training data set composed of 316 observations for 42 predictors and 1 response were used to train the model by using the method TreeBag which performs the bagging function using the Caret Package. The method was then replaced by rpart to tune the hyperparameter which is the cost complexity and results based on accuracy as a metric were obtained.



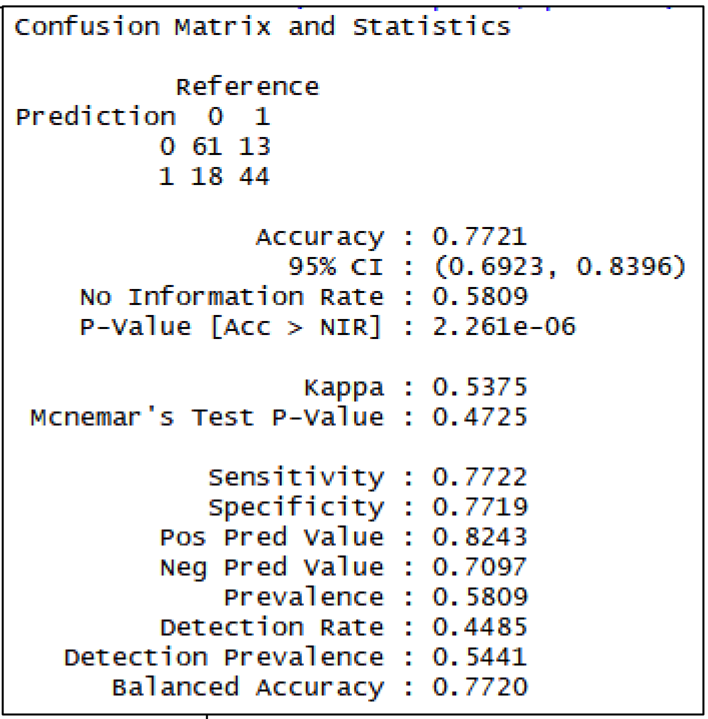
```
CART

316 samples
 42 predictor
  2 classes: '0', '1'

Pre-processing:  (None)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 285, 285, 284, 283, 285, 284, ...
Resampling results across tuning parameters:

  cp          Accuracy   Kappa
  0.08275862  0.7286321  0.44308799
  0.18620690  0.6624196  0.29253941
  0.26896552  0.5760498  0.09122464

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.08275862.
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 69  5
         1 35 27

               Accuracy : 0.7059
                 95% CI : (0.6217, 0.7809)
    No Information Rate : 0.7647
    P-Value [Acc > NIR] : 0.9543

                  Kappa : 0.3829
 Mcnemar's Test P-Value : 4.533e-06

            Sensitivity : 0.6635
            Specificity : 0.8438
         Pos Pred Value : 0.9324
         Neg Pred Value : 0.4355
             Prevalence : 0.7647
         Detection Rate : 0.5074
   Detection Prevalence : 0.5441
      Balanced Accuracy : 0.7536

       'Positive' Class : 0
```

Figure 34: Resampled parameters using bagging technique and confusion matrix

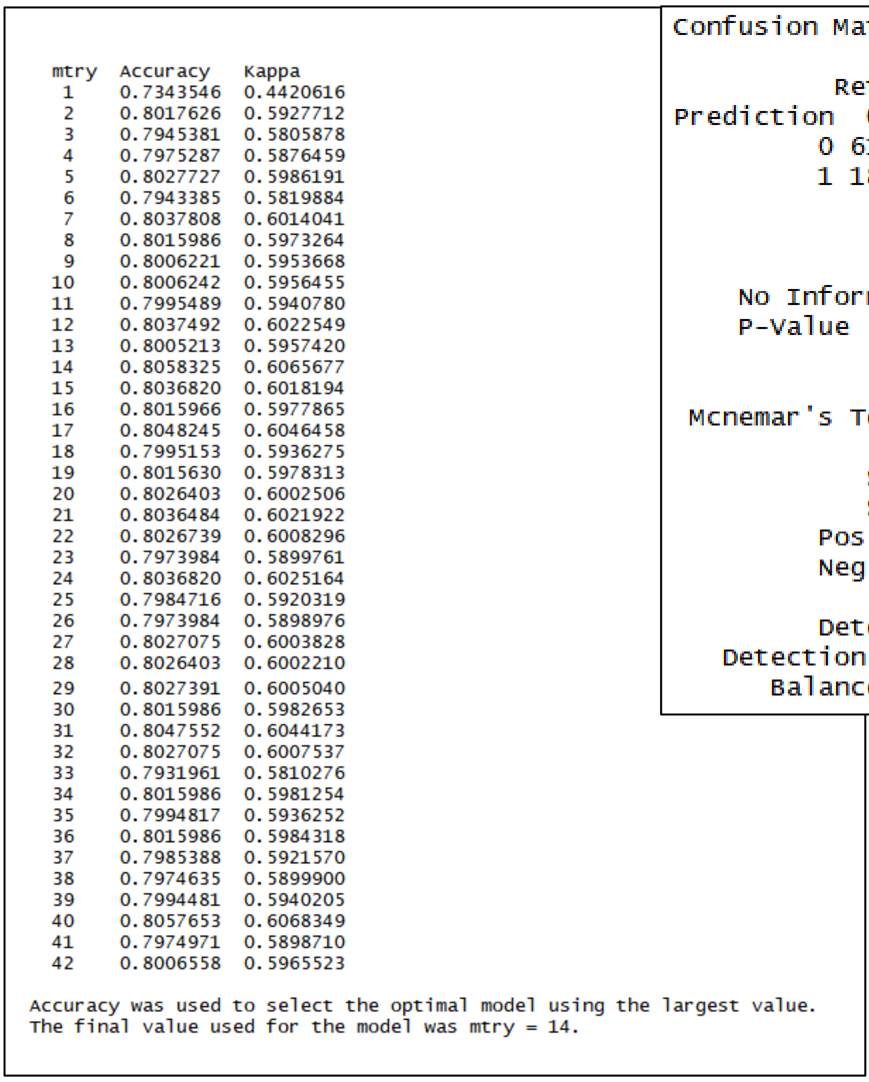
The accuracy obtained for the normal bagging model was found to 70.59%. However, upon tuning, the cost complexity parameter, an optimal model was yielded that corresponded to an accuracy of 72.86% by using the value of cost complexity parameter as 0.0827.

- <u>Random Forest</u>
  The training data set composed of 316 observations for 42 predictors and 1 response were used to train the model by using the method rf which performs the random forest search function using the Caret Package. The method was then tuned for the best value of number of trees and size of tree by using an expansive grid search and optimizing the model based on accuracy as an outcome.



```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 61 13
         1 18 44

               Accuracy : 0.7721
                 95% CI : (0.6923, 0.8396)
    No Information Rate : 0.5809
    P-Value [Acc > NIR] : 2.261e-06

                  Kappa : 0.5375
 Mcnemar's Test P-Value : 0.4725

            Sensitivity : 0.7722
            Specificity : 0.7719
         Pos Pred Value : 0.8243
         Neg Pred Value : 0.7097
             Prevalence : 0.5809
         Detection Rate : 0.4485
   Detection Prevalence : 0.5441
      Balanced Accuracy : 0.7720
```

Figure 35: Confusion matrix



```
  mtry  Accuracy   Kappa
   1    0.7343546  0.4420616
   2    0.8017626  0.5927712
   3    0.7945381  0.5805878
   4    0.7975287  0.5876459
   5    0.8027727  0.5986191
   6    0.7943385  0.5819884
   7    0.8037808  0.6014041
   8    0.8015986  0.5973264
   9    0.8006221  0.5953668
  10    0.8006242  0.5956455
  11    0.7995489  0.5940780
  12    0.8037492  0.6022549
  13    0.8005213  0.5957420
  14    0.8058325  0.6065677
  15    0.8036820  0.6018194
  16    0.8015966  0.5977865
  17    0.8048245  0.6046458
  18    0.7995153  0.5936275
  19    0.8015630  0.5978313
  20    0.8026403  0.6002506
  21    0.8036484  0.6021922
  22    0.8026739  0.6008296
  23    0.7973984  0.5899761
  24    0.8036820  0.6025164
  25    0.7984716  0.5920319
  26    0.7973984  0.5898976
  27    0.8027075  0.6003828
  28    0.8026403  0.6002210
  29    0.8027391  0.6005040
  30    0.8015986  0.5982653
  31    0.8047552  0.6044173
  32    0.8027075  0.6007537
  33    0.7931961  0.5810276
  34    0.8015986  0.5981254
  35    0.7994817  0.5936252
  36    0.8015986  0.5984318
  37    0.7985388  0.5921570
  38    0.7974635  0.5899900
  39    0.7994481  0.5940205
  40    0.8057653  0.6068349
  41    0.7974971  0.5898710
  42    0.8006558  0.5965523

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 14.
```

Figure 36: Accuracy used to select optimal model

The accuracy in the non-tuned model was obtained to be equal to 77.21%, while upon tuning the model for the best values of number of trees, it was found that the model performed the best for the number of trees as 14 (which is also equal to one-third of the number of predictors). The most optimal model had an accuracy equal to 805.8%

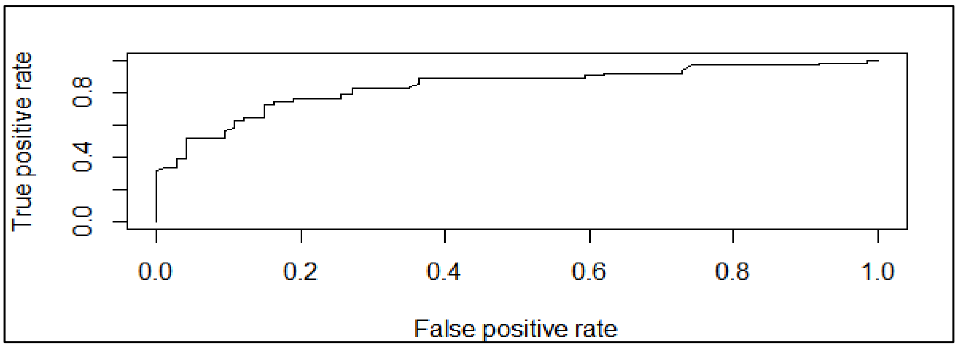
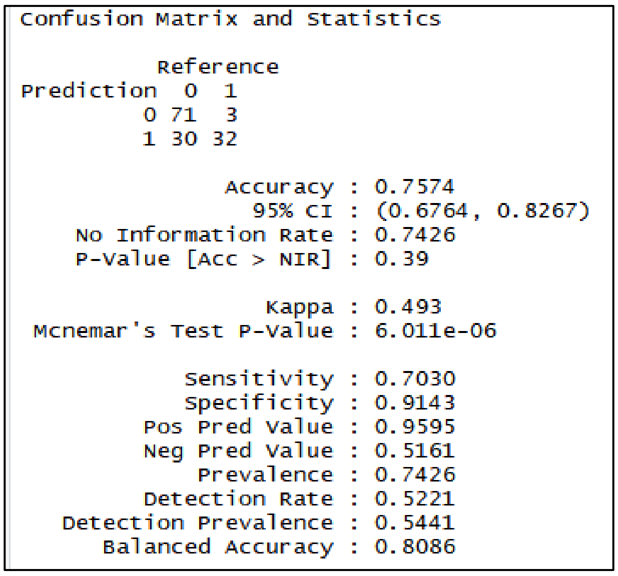


Figure 37: Plot between false and true positive rate

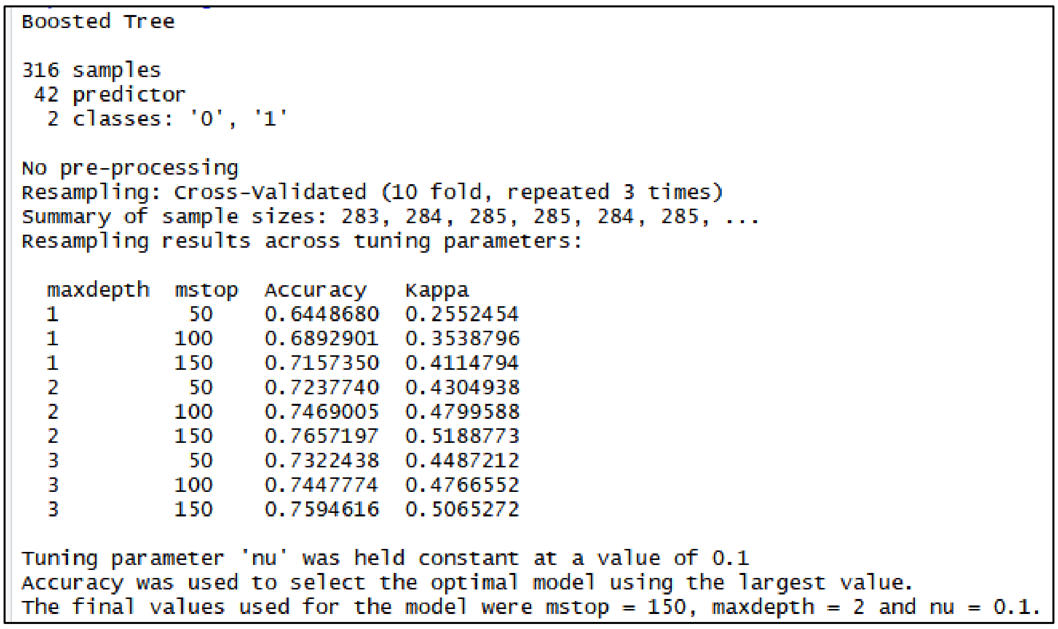
- <u>Boosting</u>

  The training data set composed of 316 observations for 42 predictors and 1 response were used to train the model by using the method bstTree which performs the boosting function using the Caret Package. The method was then tuned by using the method adaboost and an optimal model based on accuracy as a metric was obtained as an output.



```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 71  3
         1 30 32

               Accuracy : 0.7574
                 95% CI : (0.6764, 0.8267)
    No Information Rate : 0.7426
    P-Value [Acc > NIR] : 0.39

                  Kappa : 0.493
 Mcnemar's Test P-Value : 6.011e-06

            Sensitivity : 0.7030
            Specificity : 0.9143
         Pos Pred Value : 0.9595
         Neg Pred Value : 0.5161
             Prevalence : 0.7426
         Detection Rate : 0.5221
   Detection Prevalence : 0.5441
      Balanced Accuracy : 0.8086
```

Figure 38: Confusion matrix of parameters using boosting technique

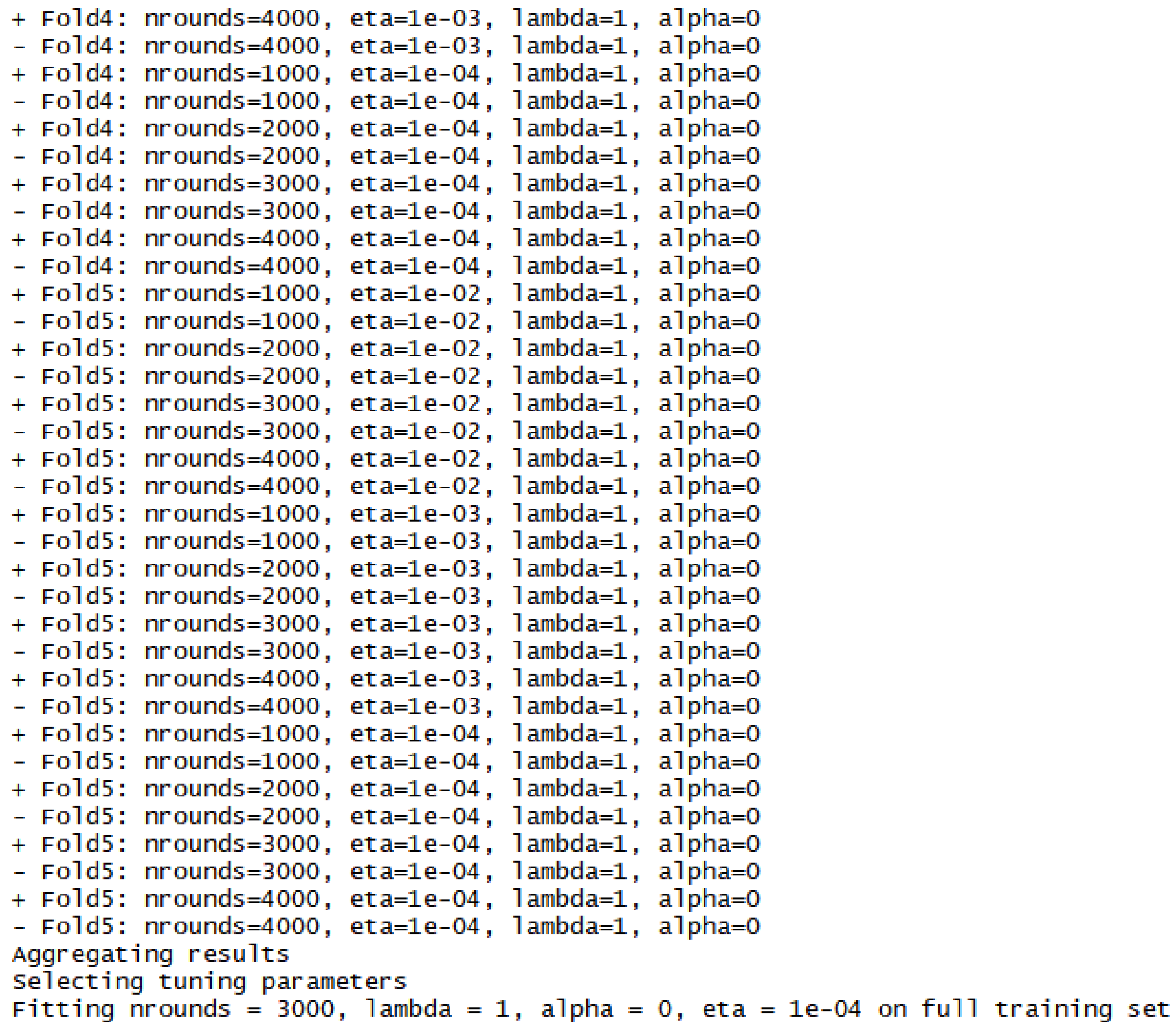


```
Boosted Tree

316 samples
 42 predictor
  2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 283, 284, 285, 285, 284, 285, ...
Resampling results across tuning parameters:

  maxdepth  mstop  Accuracy   Kappa
  1          50    0.6448680  0.2552454
  1         100    0.6892901  0.3538796
  1         150    0.7157350  0.4114794
  2          50    0.7237740  0.4304938
  2         100    0.7469005  0.4799588
  2         150    0.7657197  0.5188773
  3          50    0.7322438  0.4487212
  3         100    0.7447774  0.4766552
  3         150    0.7594616  0.5065272

Tuning parameter 'nu' was held constant at a value of 0.1
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were mstop = 150, maxdepth = 2 and nu = 0.1.
```

Figure 39:Summary of resampled data using C.V. for parameter tuning using boosting technique

The model accuracy for a non-tuned model was found to be equal to 75.74%. However, upon tuning the model for the best combination of depth, nu and mstop using an expansive grid search, it was found that the model accuracy got increased to 76.57%

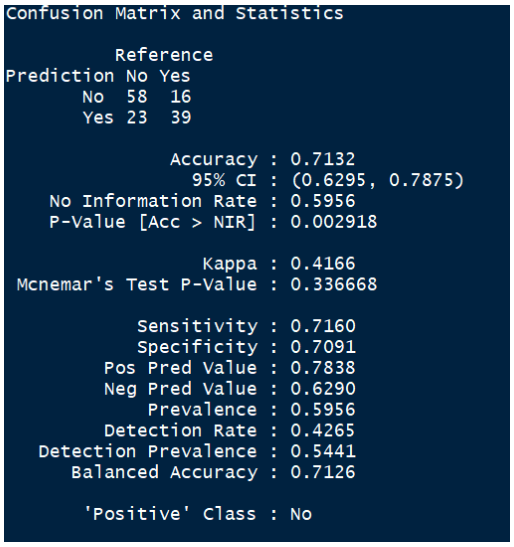
- Gradient Boosting



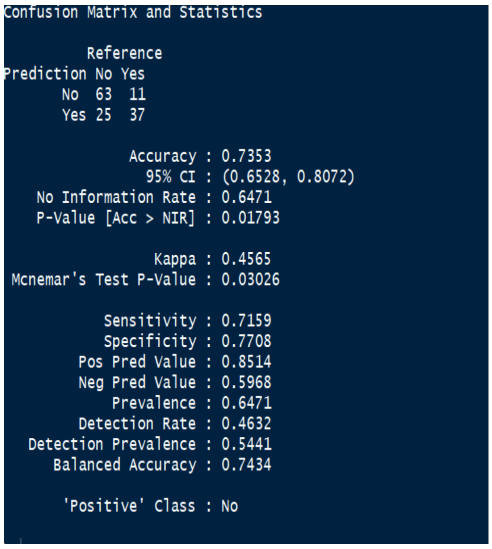
```
+ Fold4: nrounds=4000, eta=1e-03, lambda=1, alpha=0
- Fold4: nrounds=4000, eta=1e-03, lambda=1, alpha=0
+ Fold4: nrounds=1000, eta=1e-04, lambda=1, alpha=0
- Fold4: nrounds=1000, eta=1e-04, lambda=1, alpha=0
+ Fold4: nrounds=2000, eta=1e-04, lambda=1, alpha=0
- Fold4: nrounds=2000, eta=1e-04, lambda=1, alpha=0
+ Fold4: nrounds=3000, eta=1e-04, lambda=1, alpha=0
- Fold4: nrounds=3000, eta=1e-04, lambda=1, alpha=0
+ Fold4: nrounds=4000, eta=1e-04, lambda=1, alpha=0
- Fold4: nrounds=4000, eta=1e-04, lambda=1, alpha=0
+ Fold5: nrounds=1000, eta=1e-02, lambda=1, alpha=0
- Fold5: nrounds=1000, eta=1e-02, lambda=1, alpha=0
+ Fold5: nrounds=2000, eta=1e-02, lambda=1, alpha=0
- Fold5: nrounds=2000, eta=1e-02, lambda=1, alpha=0
+ Fold5: nrounds=3000, eta=1e-02, lambda=1, alpha=0
- Fold5: nrounds=3000, eta=1e-02, lambda=1, alpha=0
+ Fold5: nrounds=4000, eta=1e-02, lambda=1, alpha=0
- Fold5: nrounds=4000, eta=1e-02, lambda=1, alpha=0
+ Fold5: nrounds=1000, eta=1e-03, lambda=1, alpha=0
- Fold5: nrounds=1000, eta=1e-03, lambda=1, alpha=0
+ Fold5: nrounds=2000, eta=1e-03, lambda=1, alpha=0
- Fold5: nrounds=2000, eta=1e-03, lambda=1, alpha=0
+ Fold5: nrounds=3000, eta=1e-03, lambda=1, alpha=0
- Fold5: nrounds=3000, eta=1e-03, lambda=1, alpha=0
+ Fold5: nrounds=4000, eta=1e-03, lambda=1, alpha=0
- Fold5: nrounds=4000, eta=1e-03, lambda=1, alpha=0
+ Fold5: nrounds=1000, eta=1e-04, lambda=1, alpha=0
- Fold5: nrounds=1000, eta=1e-04, lambda=1, alpha=0
+ Fold5: nrounds=2000, eta=1e-04, lambda=1, alpha=0
- Fold5: nrounds=2000, eta=1e-04, lambda=1, alpha=0
+ Fold5: nrounds=3000, eta=1e-04, lambda=1, alpha=0
- Fold5: nrounds=3000, eta=1e-04, lambda=1, alpha=0
+ Fold5: nrounds=4000, eta=1e-04, lambda=1, alpha=0
- Fold5: nrounds=4000, eta=1e-04, lambda=1, alpha=0
Aggregating results
Selecting tuning parameters
Fitting nrounds = 3000, lambda = 1, alpha = 0, eta = 1e-04 on full training set
```

The training data set composed of 316 observations for 42 predictors and 1 response were used to train the model by using the method xgbLinear which performs the gradient boosting function using the Caret Package. The method was then tuned by using an expansive grid search consisting of number of rounds, the learning rate, lambda and alpha and finally an optimal model based on accuracy as a metric was obtained as an output.

- Neural Networks
  The training data set composed of 316 observations for 42 predictors and 1 response were used to train the model by using the method nnet which performs the Neural Network function using the Caret Package. The method was then tuned by varying the value of number of hidden layers in increments of 1 from 3 to 10 and the learning rate decay from 0.1 to 0.5 by increments of 0.1 An optimal model based on accuracy as a metric was finally obtained as an output.



```
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No  58  16
       Yes 23  39

               Accuracy : 0.7132
                 95% CI : (0.6295, 0.7875)
    No Information Rate : 0.5956
    P-Value [Acc > NIR] : 0.002918

                  Kappa : 0.4166
 Mcnemar's Test P-Value : 0.336668

            Sensitivity : 0.7160
            Specificity : 0.7091
         Pos Pred Value : 0.7838
         Neg Pred Value : 0.6290
             Prevalence : 0.5956
         Detection Rate : 0.4265
   Detection Prevalence : 0.5441
      Balanced Accuracy : 0.7126

       'Positive' Class : No
```



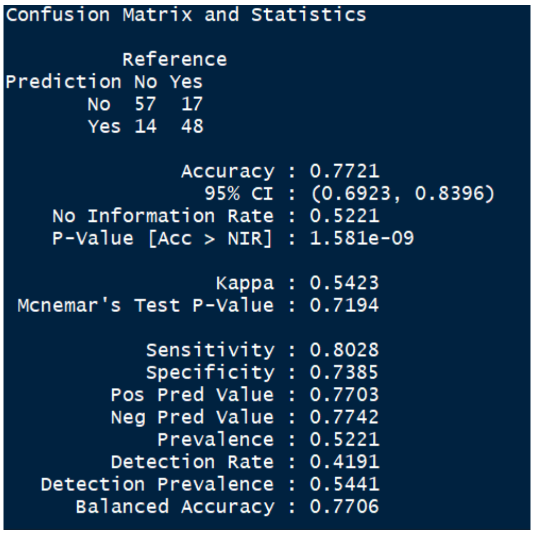
```
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No  63  11
       Yes 25  37

               Accuracy : 0.7353
                 95% CI : (0.6528, 0.8072)
    No Information Rate : 0.6471
    P-Value [Acc > NIR] : 0.01793

                  Kappa : 0.4565
 Mcnemar's Test P-Value : 0.03026

            Sensitivity : 0.7159
            Specificity : 0.7708
         Pos Pred Value : 0.8514
         Neg Pred Value : 0.5968
             Prevalence : 0.6471
         Detection Rate : 0.4632
   Detection Prevalence : 0.5441
      Balanced Accuracy : 0.7434

       'Positive' Class : No
```

Figure 40: Confusion matrix for tuned and untuned parameter evaluation using neural networks

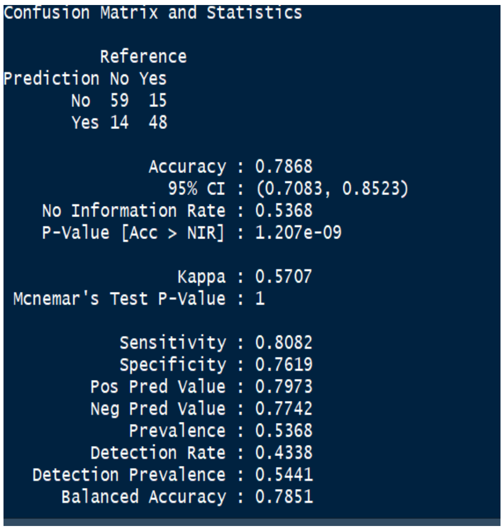
The accuracy obtained for the normal neural networks model was found to 71.32%. However, upon tuning the number of hidden layers and the decay rate, an optimal model was generated that corresponded to an accuracy of 73.53%.

- Ensemble Methods

  The training data set composed of 316 observations for 42 predictors and 1 response were used to train the model by using the method "C5.0" which performs the ensemble function using the Caret Package. The method was then tuned by varying the number of trials, number of models and winnow. An optimal model based on accuracy as a metric was finally obtained as an output.



```
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No  57  17
       Yes 14  48

               Accuracy : 0.7721
                 95% CI : (0.6923, 0.8396)
    No Information Rate : 0.5221
    P-Value [Acc > NIR] : 1.581e-09

                  Kappa : 0.5423
 Mcnemar's Test P-Value : 0.7194

            Sensitivity : 0.8028
            Specificity : 0.7385
         Pos Pred Value : 0.7703
         Neg Pred Value : 0.7742
             Prevalence : 0.5221
         Detection Rate : 0.4191
   Detection Prevalence : 0.5441
      Balanced Accuracy : 0.7706
```



```
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No  59  15
       Yes 14  48

               Accuracy : 0.7868
                 95% CI : (0.7083, 0.8523)
    No Information Rate : 0.5368
    P-Value [Acc > NIR] : 1.207e-09

                  Kappa : 0.5707
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.8082
            Specificity : 0.7619
         Pos Pred Value : 0.7973
         Neg Pred Value : 0.7742
             Prevalence : 0.5368
         Detection Rate : 0.4338
   Detection Prevalence : 0.5441
      Balanced Accuracy : 0.7851
```

Figure 41: Confusion matrix for tuned and untuned parameter evaluation using Ensemble methods

Comparison of different methods

	Training 70%, Test 30%		Training 80%, Test 20%		Training 90%, Test 10%	
Technique used	Accuracy	Accuracy post-tuning	Accuracy	Accuracy post-tuning	Accuracy	Accuracy post-tuning
Logistic Regression	71.3%	82.4%	72.5%	80.6%	76.1%	80.9%
LDA	69.9%	80.6%	72.5%	80.5%	73.9%	80.9%
QDA	70.4%	78.9%	73.1%	80.2%	74.2%	81.3%
KNN	72.8%	72.8%	67.0%	78.7%	76.1%	78.2%
SVM Radial	75.0%	82.7%	72.5%	81.4%	71.7%	81.8%
SVM Linear	73.5%	80.8%	75.8%	80.8%	71.7%	80.8%
SVM Polynomial	76.5%	81.9%	75.8%	80.8%	71.7%	80.8%
Bagging	70.6%	72.9%	64.8%	71.4%	67.4%	78.8%
Random forest	77.2%	80.6%	70.3%	81.2%	71.7%	81.3%
Boosting	75.7%	76.6%	73.6%	75.3%	73.9%	76.0%
Neural network	71.3%	73.5%	67.8%	78.5%	71.7%	76.1%
Gradient boosting	76.5%	73.5%	71.4%	71.4%	71.7%	80.8%
Ensemble (C5.O)	77.2%	78.7%	73.6%	81.8%	72.3%	78.2%

The table above summarises the results of predictive modelling incorporated upon the dataset consisting of significant predictors obtained through feature sub-selection. The major feature sub-selection techniques used were Boruta, LASSO, Principal Component Analysis and MARS and it was found that LASSO yielded the minimum number of predictors with maximum accuracy. Hence, subset selection was leveraged using LASSO and the dataset containing these predictors was subjected to the techniques mentioned above. The dataset was split into three combinations of training and testing dataset as (70:30), (80:20) and (90:10) and the model accuracy was calculated on the test data after generating the model using the respective training data. It was found that the technique of SVM Radial yielded the maximum accuracy (i.e. minimum misclassification rate) and that value was equal to 82.7% for the data split in the form of 70% training data set and 30% testing dataset.

Executive Summary

As a medical practitioner, we would like to know every time a patient is tested for arrhythmia. To ensure correct detection, we expect the model to perform true prediction so that the true positive rate be as high as possible. It is important hence to identify the predictors with highest effect on response to avoid falsely identifying an underlying condition. Here in our project, for the given data set, SVM radial method has the highest accuracy of approximately 82% as compared to our other methods.

Conclusions

For supervised concept learning methodology, classification accuracy is one of the main measures of performance. Most commonly used one is measuring the correctness of classified test data over the entire tested instances. To measure the classification accuracy, we tuned all the parameters for SVM polynomial, SVM radial, LDA, QDA, KNN and decision trees. Out of these classifiers, SVM Radial with C=0.5 and gamma=0.0324 proved to be the best classifier on the Arrythmia data set with an accuracy of 82.7% on the test data.

Future works

In our work, we have basically used 4 feature selection techniques to find out the most important predictors in our dataset namely LASSO, Boruta, PCA and MARS and have then used different classification techniques to find out the best classifier on the test data set. We have also used a simple feed forward neural network varying the number of layers from 3 to 10 and learning rate from 0.1 to 0.5. For larger computational complexities involved, we were not able to use more sophisticated deeper neural networks on this dataset like VGG16 networks or ReSNeT 51/101 networks which may take up the accuracy of our dataset to more than 90%.

References

1. *CRAN - Package Glmnet*, cran.r-project.org/web/packages/glmnet/index.html.
2. https://cran.r-project.org/web/packages/caret/caret.pdf
3. “ggplot2.” *ggplot2*, ggplot2.org/
4. “RandomForest.” *Function | R Documentation*, www.rdocumentation.org/packages/randomForest/versions/4.6-14/topics/randomForest.
5. Guvenir, H.a., et al. “A Supervised Machine Learning Algorithm for Arrhythmia Analysis.”*Computers in Cardiology 1997*, doi:10.1109/cic.1997.647926
6. Mustaqeem, Anam, et al. “Multiclass Classification of Cardiac Arrhythmia Using Improved Feature Selection and SVM Invariants.” *Computational and Mathematical Methods in Medicine*, vol. 2018, 2018, pp. 1–10., doi:10.1155/2018/7310496.
7. Vicapow. “Principal Component Analysis Explained Visually.” *Explained Visually*, setosa.io/ev/principal-component-analysis/.
8. Lever, Jake, et al. “Points of Significance: Principal Component Analysis.” *Nature News*, Nature Publishing Group, 29 June 2017, www.nature.com/articles/nmeth.4346.
9. Schoonjans, Frank. “Logistic Regression.” *MedCalc*, MedCalc Software, 20 Apr. 2017, www.medcalc.org/manual/logistic_regression.php.
10. “Boosting (Machine Learning).” *Wikipedia*, Wikimedia Foundation, 30 Apr. 2018, en.wikipedia.org/wiki/Boosting_(machine_learning)
11. “Bootstrap Aggregating.” *Wikipedia*, Wikimedia Foundation, 30 Apr. 2018, en.wikipedia.org/wiki/Bootstrap_aggregating.
12. “Random Forest.” *Wikipedia*, Wikimedia Foundation, 30 Apr. 2018, en.wikipedia.org/wiki/Random_forest.
13. “Lasso (Statistics).” *Wikipedia*, Wikimedia Foundation, 3 Apr. 2018, en.wikipedia.org/wiki/Lasso_(statistics)
14. “Quadratic Classifier.” *Wikipedia*, Wikimedia Foundation, 18 May 2017, en.wikipedia.org/wiki/Quadratic_classifier#Quadratic_discriminant_analysis.
15. “K-Nearest Neighbors Algorithm.” *Wikipedia*, Wikimedia Foundation, 30 Apr. 2018, en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.
16. Patel, Savan. “Chapter 2 : SVM (Support Vector Machine) - Theory – Machine Learning 101 – Medium.” *Medium*, Machine Learning 101, 3 May 2017, medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72.
17. “Kernel Functions-Introduction to SVM Kernel & Examples.” *DataFlair*, 12 Aug. 2017, data-flair.training/blogs/svm-kernel-functions/

Appendix

```
Data_Raw <- read.csv(file="C:/Users/saksh/Desktop/Working Data set.csv", header=TRUE)

attach(Data_Raw)

which(colMeans(is.na(Data_Raw)) > 0.01)

Data_without_big_missing <- Data_Raw[, -which(colMeans(is.na(Data_Raw)) > 0.01)]

dim(Data_without_big_missing)

Data_without_missing <- lapply(Data_without_big_missing, function(x) {

  x[is.na(x)] <- mean(x, na.rm = TRUE)

  x})

attach(Data_without_missing)

Data_no_missing_dfformat <- do.call(rbind.data.frame, Data_without_missing)

df_transposed <- t(Data_no_missing_dfformat)

df_transposed_new <- as.data.frame(df_transposed)

colnames(df_transposed_new)

class(df_transposed_new)

end_point<- ncol(df_transposed_new)

for(i in 1:end_point)

{

  klm<-names(Data_without_missing[i])

  names(df_transposed_new)[i]<-klm

}

rownames(df_transposed_new) <- c()

Trial_Dataframe<- Filter(var,Binary_Dataframe)

df_transposed_new <- Trial_Dataframe

Class_New<-c(1)

for(i in 1:nrow(df_transposed_new))

{

  abcd <- ifelse(Class[i]>1,"1","0")

  Class_New[i] <- abcd

}

class(Class_New)

Binary_Dataframe_part1 <- df_transposed_new[,-260]

Binary_Dataframe <- data.frame(Binary_Dataframe_part1, Class_New)

class(Binary_Dataframe)

class(Class_New)

names(Binary_Dataframe)

Data_Raw <- Binary_Dataframe

attach(Data_Raw)

preprocessParams <- preProcess(Data_Raw[,1:277], method=c("center","scale", "zv"))

#preprocessParams_normal <- preProcess(Data_Raw[,1:277], method=c("center","scale", "zv", "range"))

Binary_Dataframe <- transformed


set.seed(123)

index1=sample(nrow(Binary_Dataframe),nrow(Binary_Dataframe)*0.9)

train=Binary_Dataframe[index1,]

train_response=data.frame(train$Class_New)

test=Binary_Dataframe[-index1,]

colnames(test)

test_response=data.frame(test$Class_New)

par(mfrow=c(1,3))

pr.out=prcomp(train[, -260], scale=FALSE)

summary(pr.out)

biplot(pr.out, scale=0)

pr.out$sdev

pr.var=pr.out$sdev^2

pr.var

pve=pr.var/sum(pr.var)

plot(pve, xlab="Principal Component", ylab="Proportion of Variance Explained", ylim=c(0,1),type='b')

plot(cumsum(pve), xlab="Principal Component", ylab="Cumulative Proportion of Variance Explained", ylim=c(0,1),type='b')

par(mfrow=c(1,2))
```

```
fit_LS = glmnet(as.matrix(train[,-260]), train[,260], family="binomial", alpha=1)

plot_glmnet(fit_LS, "lambda", label=5)

fit_LS_cv = cv.glmnet(as.matrix(train[,-260]), as.matrix(as.numeric(train[,260])-1), type.measure="class", family="binomial", alpha=1)

plot(fit_LS_cv)

coef = coef(fit_LS_cv, s = "lambda.min")

coef_df = as.data.frame(as.matrix(coef))

lasso_out = rownames(coef_df)[which(coef_df[,1] != 0)] [-1]

lasso_out

class(Class_New)

Class_New <-as.factor(Class_New)

df_train <- train[,c("Class_New",lasso_out)]

class(test_response$test.Class_New)

colnames(test_response) <- c("Class_1")

attach(test_response)

df_test <- test[,c(lasso_out)]

dim(df_test)

dim(test_response)

View(test_response)

test_response_factored <- data.frame(lapply(test_response, as.factor))

str(test_response_factored)

trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)

metric <- "Accuracy"

#install.packages("e1071")

library(e1071)

colnames(df_train)[1] <- c("Class_1")

#install.packages("Boruta")

##Boruta Feature Selection:

set.seed(1234)

boruta_output <- Boruta(Class_New~., data = train, doTrace = 2)

print(boruta_output)

final.boruta = TentativeRoughFix(boruta.train)

print(final.boruta)

boruta_signif <- names(boruta_output$finalDecision[boruta_output$finalDecision %in% c("Confirmed", "Tentative")])

print(boruta_signif)

write.csv(boruta_signif,"123.csv")

plot(boruta_output, cex.axis=.7, las=2, xlab="", main="Variable Importance")

#MARS Feature Selection

marsModel = earth(Class_New ~ .,train) # build model

ev = evimp (marsModel)

# estimate variable importance

par(mfrow=c(1,1))

par(mar=c(1,1,1,1))

plot (ev)


# logistic

trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)

metric <- "Accuracy"

#install.packages("e1071")

library(e1071)

colnames(df_train)[1] <- c("Class_1")

df_train$Class_1 <- as.factor(ifelse(df_train$Class_1=="No", "0", "1"))

test_response <- as.factor(ifelse(test_response=="No", "0", "1"))

View(test_response)

View(df_train$Class_1)

attach(df_train)

# Tuned logistic Regression

trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)

metric <- "Accuracy"

set.seed(1)

fit.log <- train(Class_1~., data=df_train, method="glmnet", metric=metric, trControl=trainControl)

print(fit.log)

dim(df_train)
```

```
temp<-predict(fit.log,newdata = df_test)

table(temp, test_response)

mean(temp==test_response)

confusionMatrix(test_response, predict(fit.log, df_test))

#KNN (With Tuning)

trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)

metric <- "Accuracy"

set.seed(2)

fit.log <- train(Class_1~., data=df_train, method="knn", metric=metric, trControl=trainControl)

print(fit.log)

temp<-predict(fit.log,newdata = df_test)

table(temp, test_response)

mean(temp==test_response)

confusionMatrix(test_response, predict(fit.log, df_test))


#LDA

trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)

metric <- "Accuracy"

set.seed(1)

fit.log <- train(Class_1~., data=df_train, method="lda", metric=metric, trControl=trainControl)

print(fit.log)

temp<-predict(fit.log,newdata = df_test)

table(temp, test_response)

mean(temp==test_response)

confusionMatrix(test_response, predict(fit.log, df_test))


#LDA WITH TUNING

trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)

metric <- "Accuracy"

set.seed(1)

fit.log <- train(Class_1~., data=df_train, method="lda2", metric=metric, trControl=trainControl)

print(fit.log)

temp<-predict(fit.log,newdata = df_test)

table(temp, test_response)

mean(temp==test_response)

confusionMatrix(test_response, predict(fit.log, df_test))


#Removing correlated predictors in LDA

set.seed(7)

cutoff <- 0.7

end_point <- ncol(df_train)

correlations <- cor(df_train[,2:end_point])

highlyCorrelated <- findCorrelation(correlations, cutoff=cutoff)

i=1

for (value in highlyCorrelated) {

  print(names(df_train)[value])

  i=i+1

}

#create a new dataset without highly corrected features

datasetFeatures <- df_train[,-highlyCorrelated]

dim(datasetFeatures)


#SVM Radial (Tuned)

trainControl <- trainControl(method="repeatedCV", number=10, repeats=3)

metric <- "Accuracy"

set.seed(1)

fit.log <- train(Class_1~., data=df_train, method="svmRadial", metric=metric, preProc=c("zv"),trControl=trainControl)

print(fit.log)

temp<-predict(fit.log,newdata = df_test)

table(temp, test_response)
```

```
mean(temp==test_response)

confusionMatrix(test_response, predict(fit.log, df_test))

plot(fit.log)


#PLOT FOR SVM RADIAL USING GRID

grid <- expand.grid(C = c(0,0.01, 0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09, 0.1, 0.25, 0.5, 0.75,0.9))

trainControl <- trainControl(method="repeatedCV", number=10, repeats=3)

metric <- "Accuracy"

set.seed(1)

fit.log <- train(Class_1~., data=df_train, method="svmRadial", metric=metric, preProc=c("zv"),trControl=trainControl, tunegrid=grid, tuneLength=10)

print(fit.log)

plot(fit.log)

temp<-predict(fit.log,newdata = df_test)

table(temp, test_response)

mean(temp==test_response)

confusionMatrix(test_response, predict(fit.log, df_test))


#SVM Linear

grid <- expand.grid(C = c(0,0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2,5))

trainControl <- trainControl(method="repeatedCV", number=10, repeats=3)

metric <- "Accuracy"

set.seed(1)

fit.log <- train(df_train$Class_1~., data=datasetFeatures, method="svmLinear", metric=metric, preProc=c("zv"),trControl=trainControl, tunegrid=grid, tuneLength=10)

print(fit.log)

temp<-predict(fit.log,newdata = df_test)

table(temp, test_response)

mean(temp==test_response)

confusionMatrix(test_response, predict(fit.log, df_test))


#SVM Polynomial

trainControl <- trainControl(method="repeatedCV", number=10, repeats=3)

metric <- "Accuracy"

set.seed(1)

fit.log <- train(Class_1~., data=datasetFeatures, method="svmPoly", metric=metric, preProc=c("zv"),trControl=trainControl)

print(fit.log)

plot(fit.log)

temp<-predict(fit.log,newdata = df_test)

table(temp, test_response)

mean(temp==test_response)

confusionMatrix(test_response, predict(fit.log, df_test))


# Decision Trees (Tuned)

trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)

metric <- "Accuracy"

set.seed(1)

fit.log <- train(Class_1~., data=df_train, method="rpart", metric=metric, preProc=c("BoxCox"),trControl=trainControl)

print(fit.log)

temp<-predict(fit.log,newdata = df_test)

table(temp, test_response)

mean(temp==test_response)

confusionMatrix(test_response, predict(fit.log, df_test))


#General Bagging (Tuned via vars)

trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)

metric <- "Accuracy"
```

```
set.seed(2)

fit.log <- train(Class_1~., data=df_train, method="bag", metric=metric,trControl=trainControl)

print(fit.log)

temp<-predict(fit.log,newdata = df_test)

table(temp, test_response)

mean(temp==test_response)

confusionMatrix(test_response, predict(fit.log, df_test))


#Random Forest

trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)

metric <- "Accuracy"

set.seed(2)

fit.log <- train(Class_1~., data=df_train, method="rf", metric=metric,trControl=trainControl)

print(fit.log)

temp<-predict(fit.log,newdata = df_test)

table(temp, test_response)

mean(temp==test_response)

confusionMatrix(test_response, predict(fit.log, df_test))


##PLOT OF ROC Curve FOR RANDOM FORESTS

#install.packages("pROC")

library(pROC)

#install.packages("ROCR")

library(ROCR)

fit.log <- train(Class_1~., data=df_train, method="rf", metric=metric,trControl=trainControl)

print(fit.log)

temp<-predict(fit.log,newdata = df_test, type="prob")

temp_prediction <- prediction(temp[,2], test_response)

rf.performance <- performance(temp_prediction,"tpr","fpr")

results.roc <- plot(rf.performance) # Draw ROC curve.

par(mfrow=c(1,1))

plot(rf.performance, print.thres="best", print.thres.best.method="closest.topleft")

result.coords <- coords(result.roc, "best", best.method="closest.topleft", ret=c("threshold", "accuracy"))

print(result.coords)#to get threshold and accuracy


#RANDOM FOREST WITH TUNING FOR BEST VALUE OF MTRY (Takes atleast 30 mins to run)

set.seed(567)

tunegrid <- expand.grid(.mtry=c(1:42))

rf_gridsearch <- train(Class_1~., data=df_train, method="rf", metric=metric, tuneGrid=tunegrid, trControl=trainControl)

print(rf_gridsearch)

# Therefore best value of mtry=3 for RF


#Boosting

trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)

metric <- "Accuracy"

set.seed(3)

fit.log <- train(Class_1~., data=df_train, method="bstTree", metric=metric,trControl=trainControl)

print(fit.log)

temp<-predict(fit.log,newdata = df_test)

table(temp, test_response)

mean(temp==test_response)

confusionMatrix(test_response, predict(fit.log, df_test))


#BOOSTING VIA TUNING

#install.packages("fastAdaboost")

library(fastAdaboost)
```

```
trainControl <- trainControl(method="repeatedcv",
number=10, repeats=3)

metric <- "Accuracy"

set.seed(3)

fit.log <- train(Class_1~., data=df_train,
method="adaboost",
metric=metric,trControl=trainControl)

print(fit.log)

temp<-predict(fit.log,newdata = df_test)

table(temp, test_response)

mean(temp==test_response)

confusionMatrix(test_response, predict(fit.log,
df_test))

## C5.0 with tuning

set.seed(77)

C5.0Control <- trainControl(method =
"repeatedcv", number = 10, repeats = 3)

metric <- "Accuracy"

c50Grid <- expand.grid(.trials = c(1:9,
(1:10)*10),.model = c("tree", "rules"),.winnow =
c(TRUE, FALSE))

fit.c50_tune <- train(Class_1~., data=df_train,
method="C5.0", metric=metric,tuneGrid =c50Grid,
trControl=C5.0Control)

print(fit.c50_tune)

temp_C50_tune<-predict(fit.c50_tune, df_test)

table(temp_C50_tune,test_response$Class_1 )

mean(temp_C50_tune==test_response$Class_1)

confusionMatrix(test_response$Class_1,
predict(fit.c50_tune, df_test))

# C5.0 without tune

set.seed(77)

C5.0Control <- trainControl(method =
"repeatedcv", number = 10, repeats = 3)

metric <- "Accuracy"

#c50Grid <- expand.grid(.trials = c(1:9,
(1:10)*10),.model = c("tree", "rules"),.winnow =
c(TRUE, FALSE))

fit.c50 <- train(Class_1~., data=df_train,
method="C5.0",
metric=metric,trControl=C5.0Control)

print(fit.c50)

temp_C50<-predict(fit.c50, df_test)

table(temp_C50,test_response$Class_1 )

mean(temp_C50==test_response$Class_1)

confusionMatrix(test_response$Class_1,
predict(fit.c50, df_test))

#Neural network without tuning

set.seed(746)

NeuralControl <- trainControl(method =
"repeatedcv", number = 10, repeats = 3)

metric <- "Accuracy"

tune = expand.grid(size = seq(from = 3, to = 10, by
= 1),decay = seq(from = 0.1, to = 0.5, by = 0.1))

Neuralmodel_tune <- train(Class_1 ~ ., data =
df_train, method='nnet', trControl =
NeuralControl,metric = metric, tuneGrid =
tune,trace = FALSE)

print(Neuralmodel_tune)

temp_neural<-predict(Neuralmodel_tune, df_test)

table(temp_neural, test_response$Class_1)

mean(temp_neural==test_response$Class_1)

confusionMatrix(test_response$Class_1,
predict(Neuralmodel_tune, df_test))

#Neural Network with tuning

set.seed(746)

NeuralControl <- trainControl(method =
"repeatedcv", number = 10, repeats = 3)

metric <- "Accuracy"

#tune = expand.grid(size = seq(from = 3, to = 10, by
= 1),decay = seq(from = 0.1, to = 0.5, by = 0.1))

Neuralmodel <- train(Class_1 ~ ., data = df_train,
method='nnet', trControl = NeuralControl,metric =
metric,trace = FALSE)

print(Neuralmodel)

temp_neural<-predict(Neuralmodel, df_test)

table(temp_neural, test_response$Class_1)

mean(temp_neural==test_response$Class_1)

confusionMatrix(test_response$Class_1,
predict(Neuralmodel, df_test))
```