Individual Coursework Submission Form

Specialist Masters Programme

Surname: Parashar	**First Name: Tridev**
MSc in: Business Analytics	**Student ID number: 180032806**
Module Code: SMM 636	
Module Title: Machine Learning	
Lecturer: Dr. Rui Zhu	**Submission Date: 24 March 2023**

Declaration:
By submitting this work, I declare that this work is entirely my own except those parts duly identified and referenced in my submission. It complies with any specified word limits and the requirements and regulations detailed in the coursework instructions and any other relevant programme and module documentation. In submitting this work, I acknowledge that I have read and understood the regulations and code regarding academic misconduct, including that relating to plagiarism, as specified in the Programme Handbook. I also acknowledge that this work will be subject to a variety of checks for academic misconduct.

We acknowledge that work submitted late without a granted extension will be subject to penalties, as outlined in the Programme Handbook. Penalties will be applied for a maximum of five days lateness, after which a mark of zero will be awarded.

Marker's Comments (if not being marked on-line):

Deduction for Late Submission: | **Final Mark:** **%**

Question 1:

For this problem set, the *GermanCredit* dataset is utilized to implement the decision tree and random forest algorithms. The dataset consists of 1000 credit records, with 700 records labelled as "good" and 300 records labelled as "bad". There are 61 predictors included in the original dataset, which are related to various attributes of individuals. Upon closer examination, it was discovered that two of these predictors, *Purpose.Vacation* and *Personal.Female.Single*, have identical values for both classes. As a result, these two predictors were removed from the dataset, leaving 59 predictors to be used in the classification task. The classification model was trained using 70% of the data, and the remaining 30% was used to test the final model.

1.1 Decision Tree:

The *rpart* function from the *rpart* package is used to train a decision tree model with *5-fold* cross-validation. The *rpart.control* parameters specify that cross-validation should be performed using *5 folds*, the minimum number of instances in a terminal leaf should be 2 (*minbucket=2*), and the complexity parameter threshold should be 0.01 (*cp=0.01*). The cross-validation error rates, which are relative, are plotted against tree sizes and cp values in Figure 1. The *cptable* output of the fit contains the cross-validation error rates and indicates that the optimal cp value for pruning the tree is the one for which the error rate is below the horizontal line that is 1 standard error above the minimum error. As a result, a cp value of *0.024* is chosen to prune the tree, which has *6* terminal leaves and produces the minimum cross-validated error.

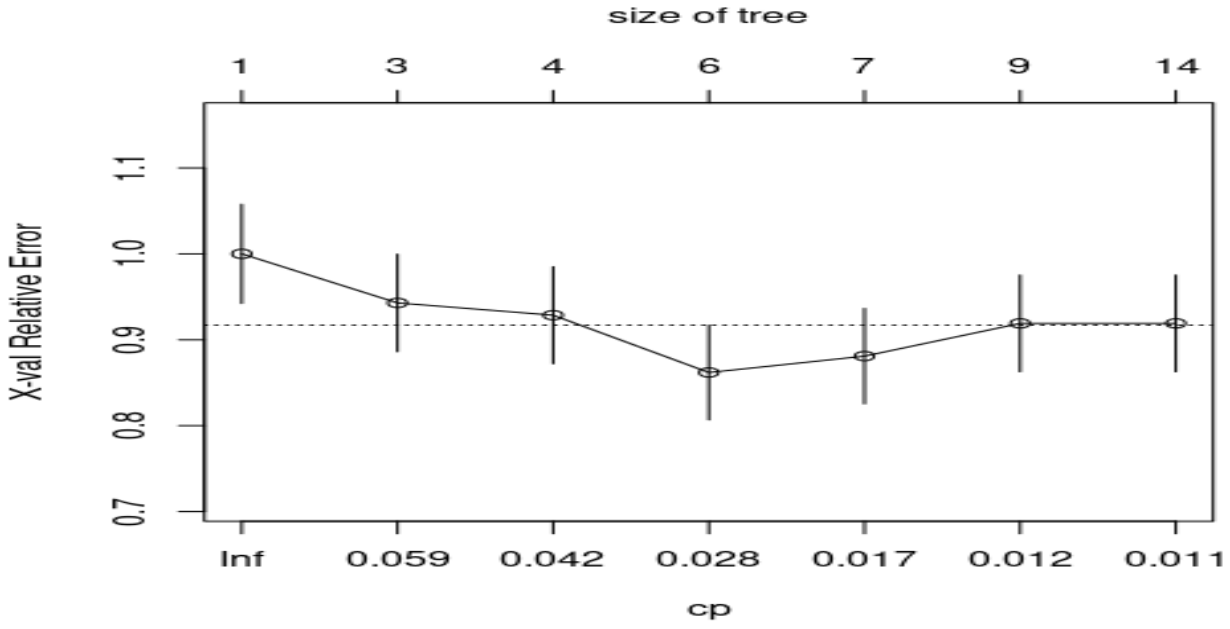

Figure 1 – Plot of cp against tree sizes

Figure 2 displays the pruned tree utilizing the *rpart.plot* package. The binary classification model indicates a green node, where most instances are classified as *Good*, while a red node signifies the majority as *Bad*. The root node contains 70% Good instances, and the initial predictor to split the tree

is *CheckingAccountStatus.none*. Each split divides instances with a yes answer to the left branch and a no answer to the right branch. The tree undergoes 5 splits, leading to 6 terminal leaves. Darker colours on the leaves indicate a higher probability of the majority class, representing a "purer" node. The purest Good node accounts for 40% of training instances, 88% of which are Good classes. In contrast, the purest Bad node contains only 2% of the observations and has an 82% probability of being classified as Bad. The results suggest that individuals without checking account status tend to have a positive credit history. However, young people (under 26 years old) with a history of unpaid credits or not paying in this bank are more likely to have a negative credit record.

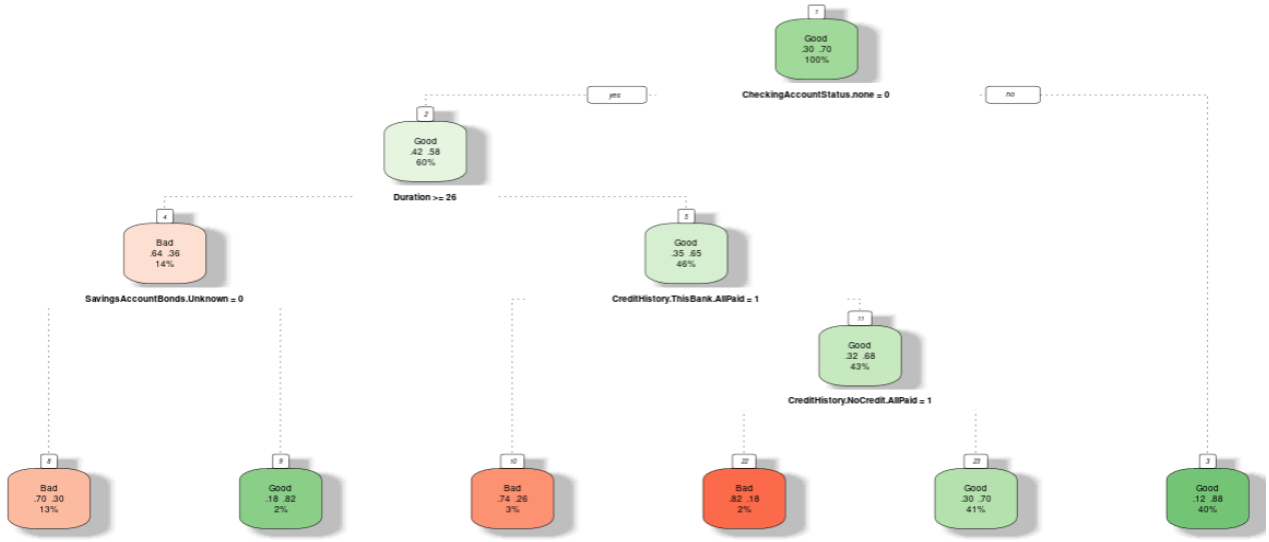

Figure 2 – The pruned decision tree of 6 terminal leaves

After implementing the pruned tree model on the test set, the obtained test error is *0.297*, which is greater than the training error of *0.224*. This result implies that the decision tree model performs marginally better on the training data than the test set, suggesting a possible overfitting problem with the decision tree algorithm. To address this issue, methods such as bagging and random forest can be utilized to enhance the model's generalization ability.

1.2 Random Forest:

The tuning of random forest models using 5-fold cross-validation is performed in the *randomForest* package, with a focus on the *mtry* parameter that determines the number of randomly sampled variables at each split. The *mtry* parameter has been tuned utilizing the *tuneRF* method, with a *ntree* value of 1000. The final model selected is *mtry=10*, which achieves the highest classification accuracy of *75.5%* on the training set, as seen in the confusion matrix below in Figure 3.

	Bad	Good
Bad	81	129
Good	43	447

Figure 3 – Confusion Matrix for the final random forest model.

The test data is classified using the final random forest model, resulting in a corresponding error rate of *0.253*. This error rate is lower than the decision tree test error rate of *0.297*. This implies that the random forest algorithm has slightly enhanced the classification accuracy of the model on the test set. Random forest is commonly employed to address the overfitting issue of decision tree models and to increase the generalization ability of the model.

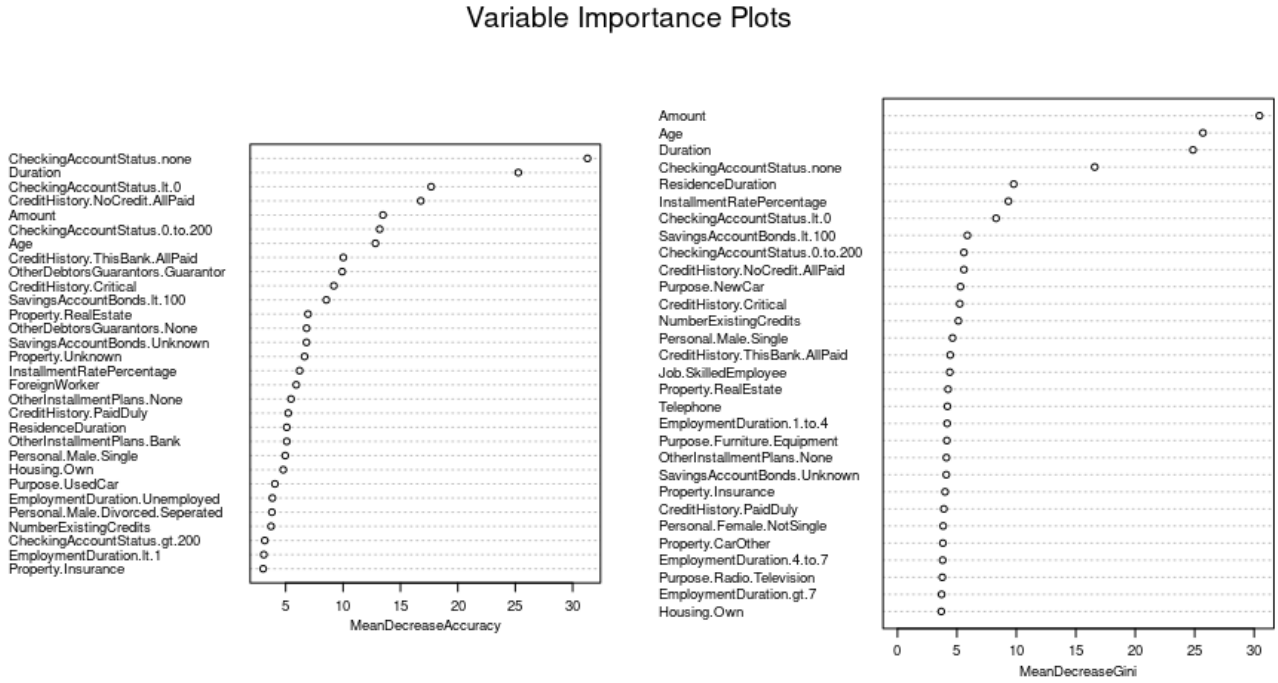

Figure 4 – Variable Importance Plots

Mean Decrease Accuracy and Mean Decrease Gini Coefficient methods are used to evaluate the importance of variables in predicting the target variable. Figure 4 displays the variable importance assessed by these methods. The larger the values of the variables, the more important they are in making predictions. The plot on the left, which depicts the Mean Decrease Accuracy values, measures how much the accuracy decreases or changes when a specific variable is excluded. *Duration*, *CheckingAccountStatus.lt.0*, and *CheckingAccountStatus.none* are the three most important variables if the top three are selected. The plot on the right, which illustrates the Mean Decrease Gini values, assesses the gain or decrease in purity when a variable is split. If the top three variables are selected again, *Amount*, *Duration*, and *Age* are the most important. However, *Duration*, *CheckingAccountStatus.lt.0*, and *CheckingAccountStatus.none* are the most important variables

common to both plots. This indicates that these variables are the most crucial in making predictions and can be retained for better model fitting.

1.3 ROC Curves:

Figure 5 displays two ROC curves depicting the classification results on the test set using the decision tree and random forest models tuned as described earlier. The random forest model outperforms the decision tree model, exhibiting a larger area under the ROC curve (AUC) of 0.79, whereas the decision tree model produces an AUC of 0.70. The observed performance gain of the random forest model in classifying the test data aligns with the lower test error rate computed previously. In conclusion, the random forest algorithm yields a slight improvement in test set classification, corroborating the notion that it can address the overfitting issue of decision tree models.

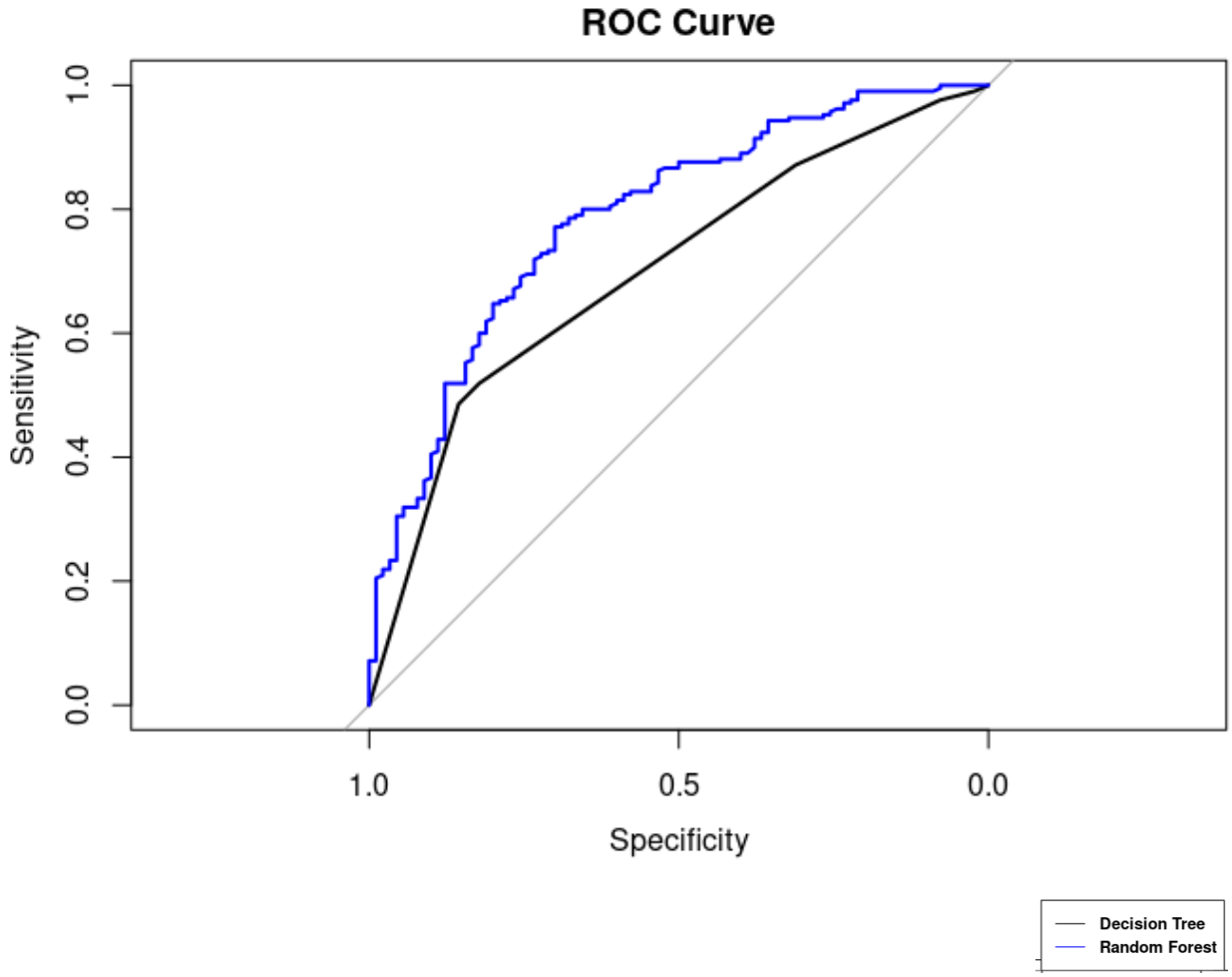

Figure 5 – ROC Curves of both decision tree and random forest

Question 2:

The process of generating a three-class dataset with 50 observations in each class and 2 features involves using the *rnorm()* function to create three separate matrices with mean zero and standard deviations of *0.2, 0.3, and 0.45* respectively. This ensures that the classes are not linearly separable. The classes are then labeled as *1, 2, and 3* and combined into a single data frame using a combination of the *rbind()* and *cbind()* functions. The column names are also changed for simplicity.

It can be observed from Figure 6 that the various classes are not easily separable in a linear manner, since all three classes seem to follow a circular or elliptical pattern. This indicates that the objective of creating a non-linearly separable dataset has been successfully accomplished. Subsequently, the data is divided into training and testing sets, with 50% of the total observations allocated to each set.

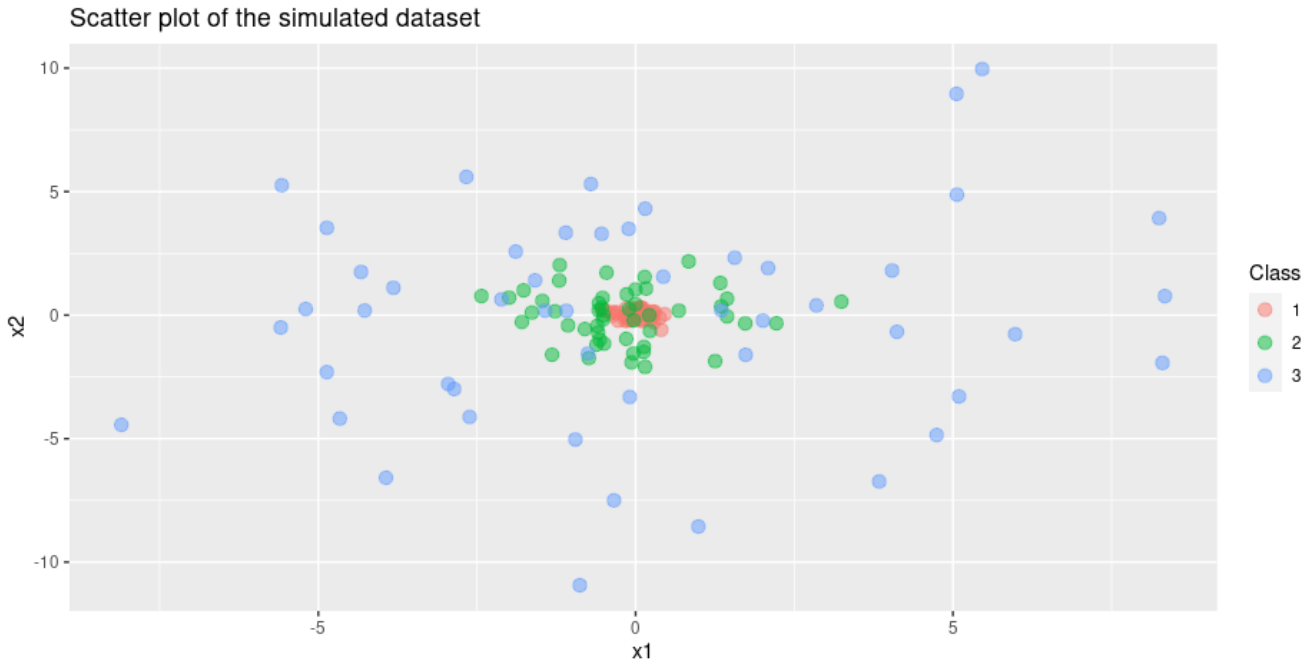

Figure 6 – Scatter plot of the simulated dataset

RBF Kernel:

Kernel functions are primarily used to solve non-linear tasks using a linear classifier, achieved through transforming the training data and calculating the inner product between two observations in a standard feature space. The Radial Basis Function (RBF) is a kernel function that depends only on the distance from the origin, making it a popular choice. It has similarities to the k-nearest neighbours method and has the added advantage of only requiring support vectors to be stored during training, instead of the whole dataset, thus reducing space complexity.

In order to use the RBF Kernel on the simulated dataset, the models' parameters are first fine-tuned using 5-fold cross-validation and the data is also scaled to ensure extreme means or standard deviations do not impact the results. The RBF Kernel produces the following results, where *C* and *sigma/gamma* are tuning parameters that determine the degree to which margin and hyperplane violations are tolerated. In theory, C balances the trade-off between bias and variance; a high value of C results in more bias but less variance, producing a wider margin and more observations that violate the margin. Conversely, a small value of C yields less bias but greater variance. Generally, there are more support vectors when C is high. However, it should be noted that when using the caret library (as well as most other R libraries), a large C value results in a narrower margin, which contradicts the

theoretical aspect. When fine-tuning sigma and C, a grid is used, where sigma and C are assigned the values of *0.01, 0.1, 1, 2, 4, 6, 8 and 10* (for sigma/gamma) and *0.01, 0.1, 1, and 10* (for C)

According to Figure 7, the sigma/gamma value of 2 and C value of 1 are selected as the optimal values since they exhibit the highest accuracy. To assess the model's performance on the test data, the test error is calculated. The resulting mean accuracy of *0.853* and test error rate of *0.147* indicate that the RBF Kernel performs effectively and generates reliable predictions.

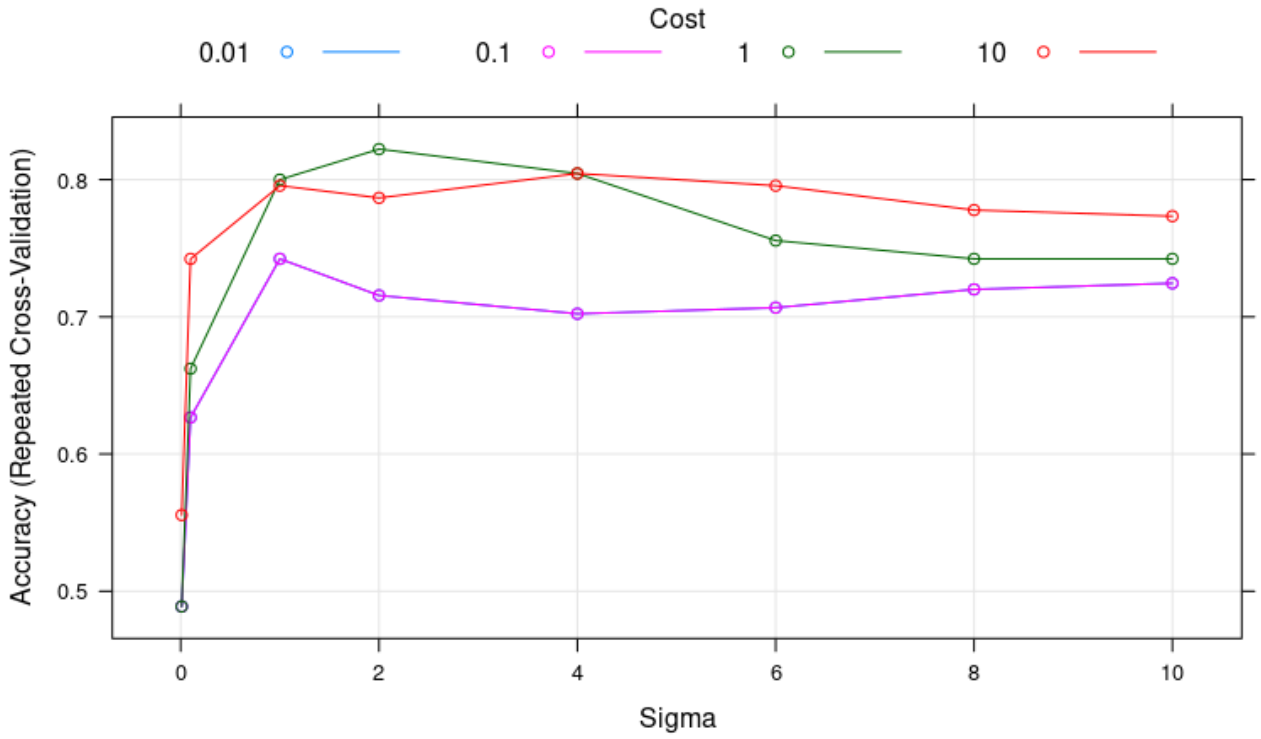

Figure 7 – Plot of training accuracy against tuning parameters for SVM with RBF kernel

Polynomial Kernel:

The SVM model with polynomial kernel involves training three parameters, namely *degree, scale, and C*. The caret package is utilized with the method '*svmPoly*' to train the model. Degree represents the degree of the polynomial, scale is the scaling parameter, and C is the cost of constraints violation. To find the optimal parameters, a grid of values is created, including *degree=c(2,3,4,5), scale=c(0.01,0.1,1), and C=c(0.01,0.1,1,10)*, and the training accuracy is plotted against different parameter values in Figure 8. Generally, a higher scaling value leads to higher accuracy. Also, a larger cost, which penalizes samples inside the margins more, tends to improve accuracy, except when scale=1. Additionally, higher polynomial degrees seem to result in higher accuracy, except when scale=1, where the impact of polynomial degree is relatively small. The optimal model is chosen based the parameters degree=2, scale=1, and C=10.

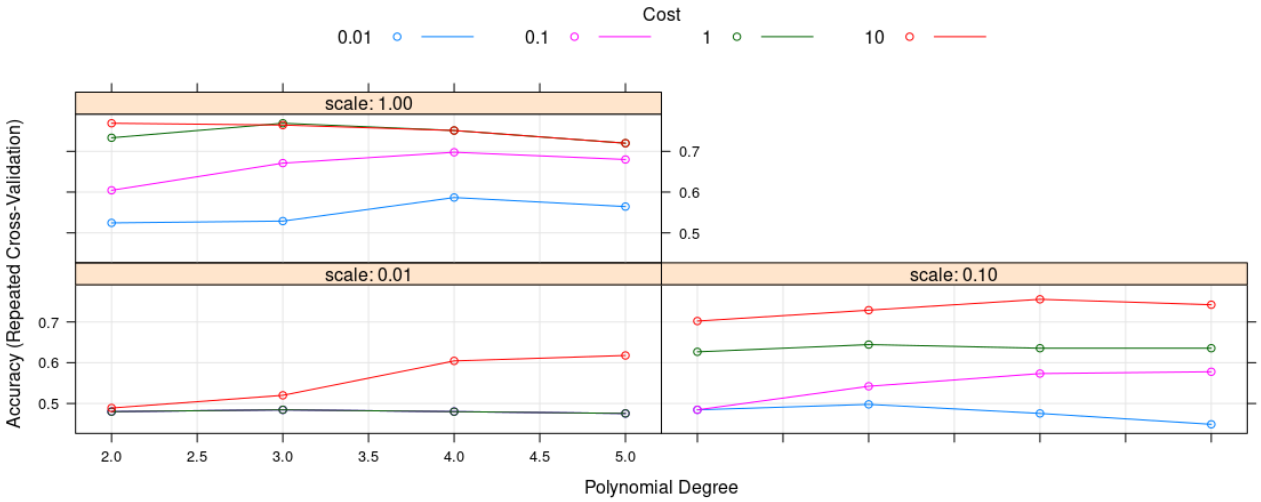

Figure 8 – Plot of training accuracy against tuning parameters for SVM with polynomial kernel

Furthermore, the mean accuracy and test error rate were evaluated, and the obtained values were *0.84* and *0.16*, respectively. These results are similar to those achieved by the RBF Kernel, with a minor decrease in mean accuracy and an increase in test error rate.

SVM with Linear Kernel (Support Vector Classifier):

The SVC, or Support Vector Classifier, extends the maximal margin classifier to handle non-separable cases. Unlike the maximal margin classifier, the SVC allows for misclassification of some training observations in order to define a hyperplane that does not perfectly separate classes. This approach may seem counterintuitive, but it can lead to greater robustness and better classification of most training data. Moreover, the SVC uses Pearson correlation to measure pairwise observation similarity. The Linear Kernel is essentially the same as the SVC, which is why the '*svmLinear*' method is used in the training process. When applying the Linear Kernel to the German Credit data, a similar approach to the RBF Kernel is taken, where a grid of *C values (0.01, 0.1, 1, 10, 20, 30, 40 and 50)* is defined. The data is also scaled/tuned within the *train ()* function, and the results are presented below.

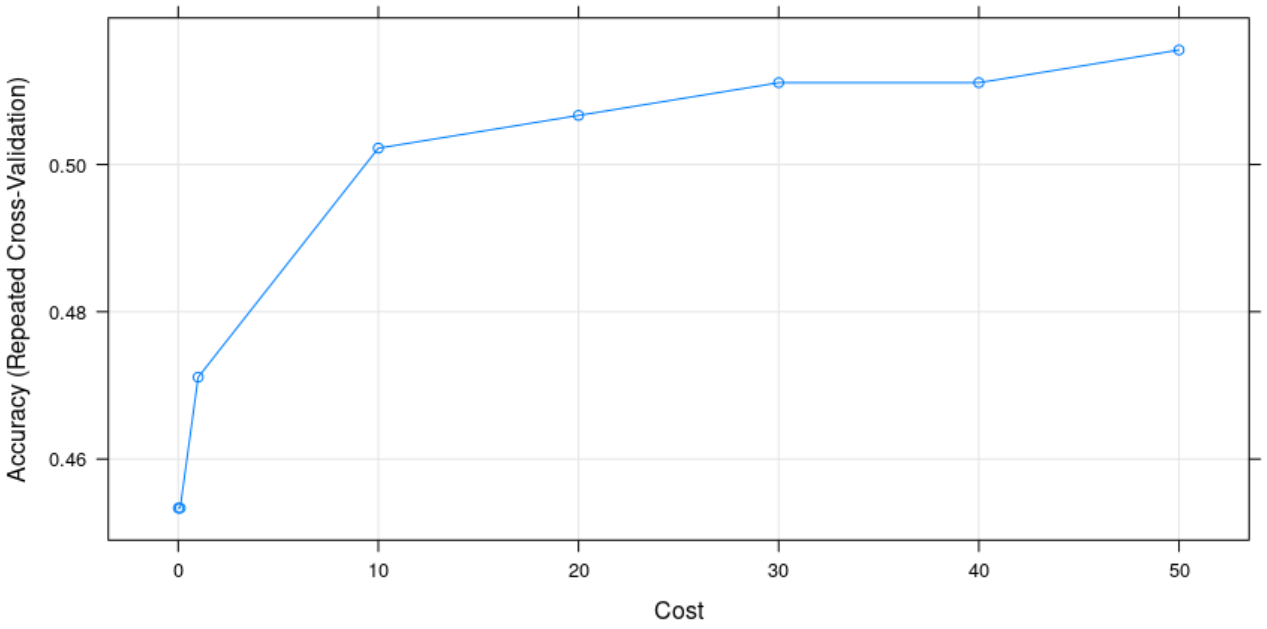

Figure 9 – Plot of training accuracy against tuning parameters of SVM with Linear Kernel

According to Figure 9, the optimal value chosen for the model was C=50. When testing the model on the test data, the mean accuracy was *0.547* and the test error rate was *0.453*. These values are relatively worse than those produced by the RBF Kernel and Polynomial Kernel SVM models, implying that the RBF Kernel performed the best among all three kernels when tested on the test data.

Question 3:

3.1 Data Exploration and AUC:

The newthyroid dataset comprises 185 samples classified into two categories - *normal patients (n) and patients with hyperthyroidism (h)* - based on five numeric features. To analyse the distribution of samples for each feature, boxplots can be utilized. Figure 10 demonstrates the five pairs of boxplots representing all the features associated with the two categories.

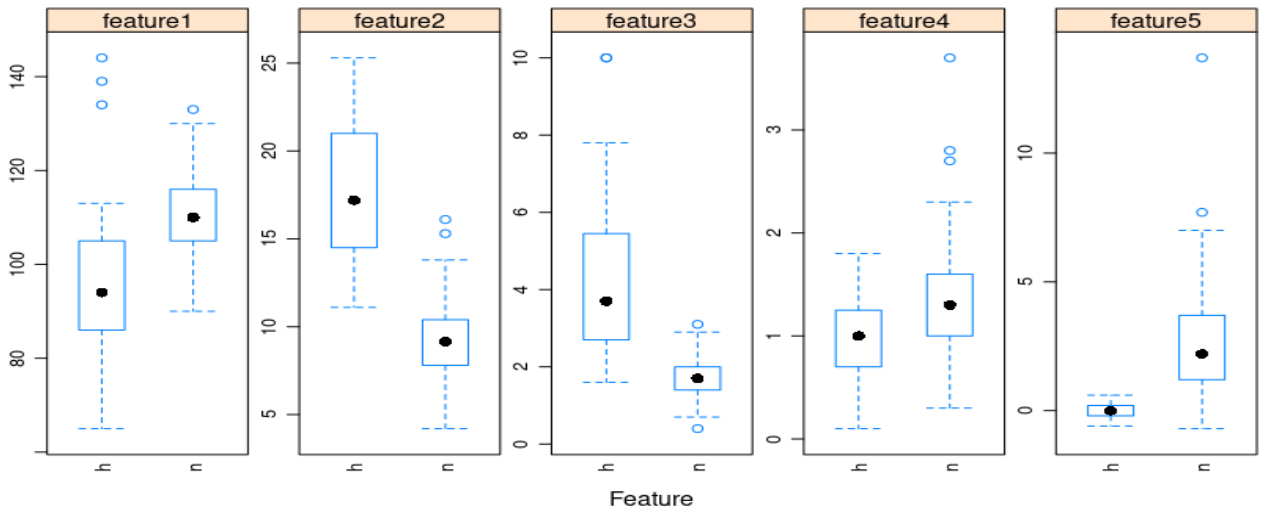

Figure 10 – Boxplots of features in the newthyroid dataset

The boxplots suggest that all five variables impact the classification of patients' categories as the medians of each feature differ for the different categories.

Prior to applying kNN and LDA to classify the dataset, random sampling is used to divide the dataset into training and test sets. This random split is repeated ten (10) times to create ten distinct training and test sets.

A for-loop is then implemented to apply kNN and LDA to each training set, and the AUC value is recorded for predicting on the associated test set. To optimize the k values for KNN, functions from the caret package are utilized, and the AUC metric is used to select k values from *(3,5,7,9,11,13,15)* based on a 5-fold cross-validation method performed three times. The k value that results in the best AUC for each random split is chosen. For LDA, since there are only two categories for each training set, only one linear direction is obtained, and functions from caret are utilized to conduct the analysis.

The *roc()* function from the *pROC* package is used to create a ROC curve, from which the AUC can be extracted. Two vectors are utilized to store the AUC values for both kNN and LDA methods in each iteration. The AUC values for all 10 random splits are presented below:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
kNN	1	1	0.998	1	1	1	1	1	1	0.996
LDA	1	1	0.995	1	1	1	1	0.997	1	1

To facilitate an assessment of both models, the ROC curves obtained from both methods are provided below in figure 11. The two curves overlap, indicating that the AUC value obtained from both methods is equal to 1, indicating an equivalent area under the curve.

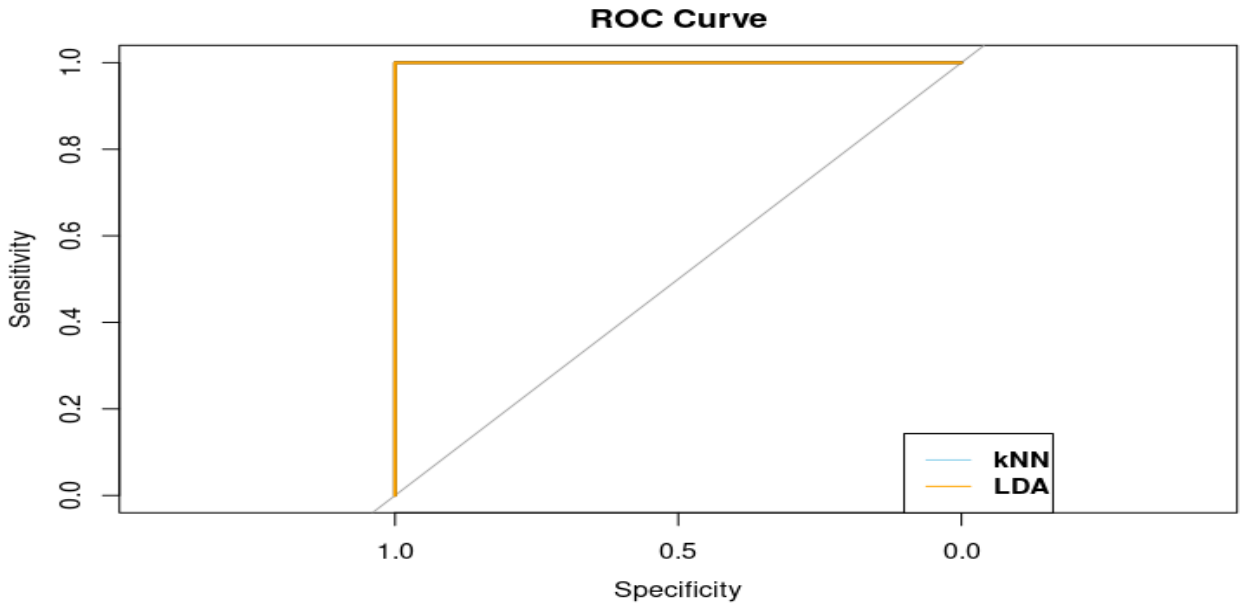

Figure 11 – ROC Curve for both kNN and LDA methods

3.2 Boxplots of both the methods:

Figure 12 displays two boxplots for kNN and LDA based on the 10 AUC values computed from 10 random splits. For the kNN classifier, the AUC values have a median close to 1 with a few outliers in the vicinity of *0.998 and 0.996*, respectively. For the LDA classifier, the AUC values have a median equal to 1 with a couple of outliers at *0.997 and 0.995*, respectively. Although the medians are comparable across both methods, the whiskers of the LDA method extend further due to the outliers having a greater range than the kNN method. This disparity could result from the training process in which AUC-based cross-validation was employed to tune the k value. The training process, which selects the optimal k value, may lead to a narrower range for the kNN boxplot.

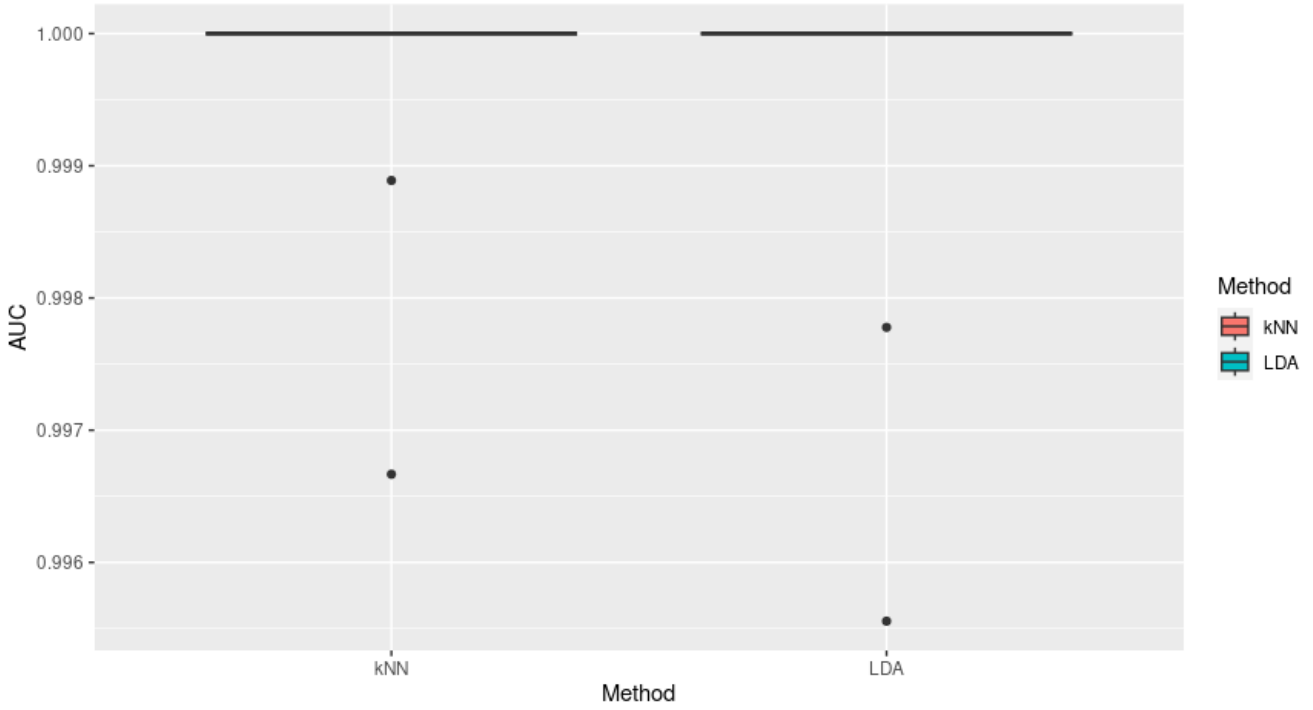

Figure 12 – Boxplots of both the kNN and LDA methods (Medians are 1 since majority of the AUC values in random splits are 1)

3.3 Conclusions

According to the analysis above, both methods demonstrate high accuracy in classifying patients in the newthyroid dataset (*LDA: 0.93 and KNN: 0.91*). However, it is crucial to consider sensitivity as an important metric, especially in cases where identifying patients with hyperthyroidism as normal could cause significant concerns. Therefore, it is essential to evaluate the sensitivity provided by both methods, using the AUC metric. Sensitivity is defined as –

$$Sensitivity = \frac{TP}{TP + FN}$$

Sensitivity for the methods under AUC metric are *0.500 (kNN)* and *0.600 (LDA)*. Since we observe that the methods under AUC metric have a high classification accuracy and AUC but a relatively lower sensitivity, we shall attempt to tune the value according to sensitivity metric rather than AUC metric.

For comparison purposes, we shall only tune the value of kNN now. Upon tuning the values based on sensitivity metric, there is a big improvement in the kNN method in comparison to the AUC metric. The sensitivity metric kNN achieves higher classification accuracy (*0.982*) and a sensitivity of *0.9*, indicating that the following misclassifies only one patient with hyperthyroidism as normal, whereas the AUC metric would classify 50% of hyperthyroidism patients as normal.

To sum up, it is crucial to choose the appropriate evaluation metrics to assess the classification performance based on the nature of the dataset. In this instance, sensitivity would be a more appropriate metric to use while training the classification model and evaluating the predicted outcomes, as incorrectly classifying patients with hyperthyroidism as normal could have serious implications in the disease classification task.

Question 4:

By utilizing Fisher's Linear Discriminant Analysis, it is possible to compute the optimal value of w by solving a problem that involves maximizing $\frac{wTSBw}{wTSWw}$. The task involves determining a direction w that maximizes the ratio of between-class scatter to within-class scatter, which can be achieved through solving an equivalent problem. By solving this problem, the direction of class mean difference normalized by within-class scatter S_w can be obtained, as depicted in equation 1. Equation 2 is utilized to calculate within-class scatter S_w through the covariance matrix. These equations are employed in a custom function (*myFDA*) to generate the linear discriminant for binary classification problems.

$$w = Sw^{-1} (\mu_1 - \mu_2) \text{ ----}(1)$$

$$\mathbf{S}_W = \sum_{c=1}^{2} \sum_{y_i=c} (\mathbf{x}_i - \boldsymbol{\mu}_c)(\mathbf{x}_i - \boldsymbol{\mu}_c)^T \quad \text{------}(2)$$

Post defining the *myFDA* function, as per the rules stated above, we applied the function on the *GermanCredit* dataset. We were unable to generate the fisher discriminants since the data corresponds to a singular matrix i.e., an inverse matrix cannot be generated (as required in equation 1 above).

The following could be possible in cases wherein the features of a dataset are correlated to each other i.e. there is presence of multicollinearity. Hence Figure 13 given below illustrates the presence the of multicollinearity in the GermanCredit dataset.

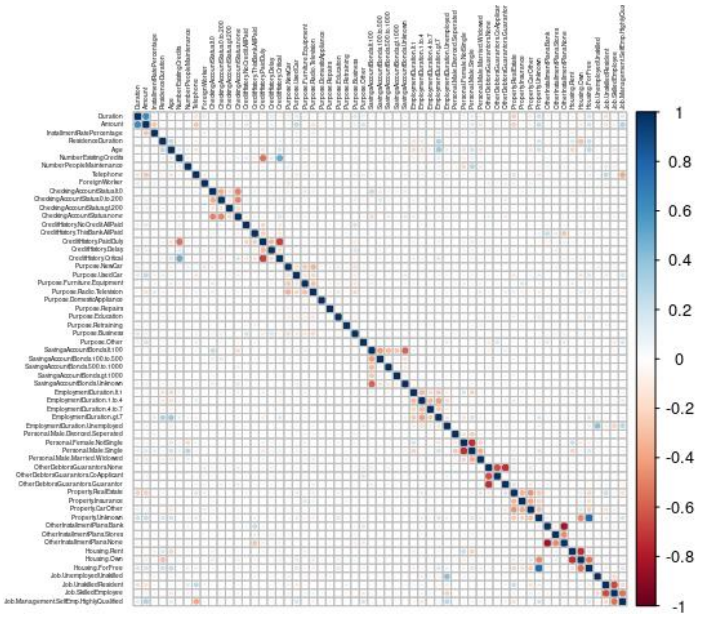

Figure 13 – Correlation Matrix for the features in GermanCredit Dataset (the red shades present along the grids indicate correlation between the features)

In order to overcome such multicollinearity, we employed PCA (dimension reduction) to mitigate such multicollinearity. We used the *'prcomp'* function to generate the principal components. Figure 14 below shows that at a level of *45* principal components (PCs), the cumulative variance startens to flattens which means that 45 PCs should be able to explain the maximum variance in the dataset without the need for any PCs.

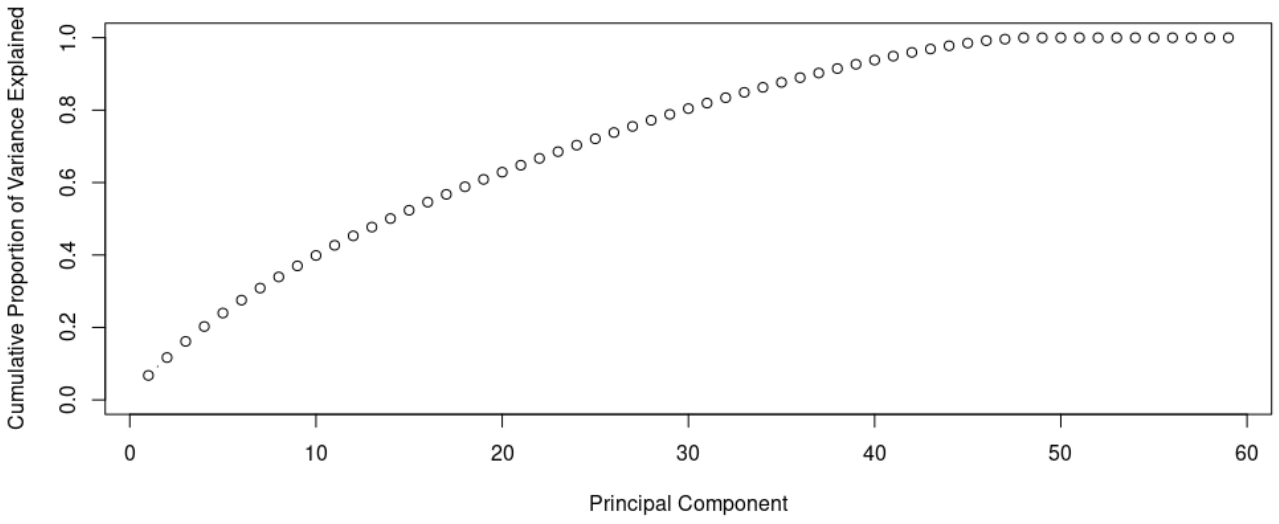

Figure 14 – Cumulative Variance Plot of the principal components generated through PCA

Thereafter, we multiply the loadings of the principal components to the training dataset and apply the myFDA function to get the fisher discriminant value.

The fisher discriminant vector generated gives us an accuracy of *52.33%* on the test dataset predictions. In order to check whether the fisher discriminant can classify the groups accurately, we further did a comparison of the myFDA function with LDA function to check for any similarity in

classification and differences in accuracy. Firstly, in order to check for similarity, we used *cosine* similarity wherein we obtained a score of 1, suggesting that the myFDA and LDA functions are very similar in this case. Since both the functions are similar, we further went ahead and checked for accuracy on the test predictions using an LDA function., which increased the accuracy to *59%*. Hence, we observe that, in this example, both the functions are similar and can be used interchangeably to fit the principal components and achieve a discriminant value for classifying the groups.