MA321-7-SP Assignment

University of Essex

Department of Mathematics, Statistics and Actuarial Science

RISHWANTH MITHRA - 2311566

Abstract:

This paper provides a comprehensive overview of Applied Statistics, and also explores both unsupervised and supervised dimension reduction techniques applied to a dataset containing 2000 observed gene expression values. Unsupervised learning approaches, such as Principal Component Analysis (PCA), k-means clustering, and hierarchical clustering, are used to identify groupings in the gene expression dataset.

Furthermore, supervised learning models are then applied, such as Logistic Regression, Linear Discriminant Analysis (LDA), k-nearest Neighbours (k-NN), Random Forest, Naïve Bayes, QDA and Support Vector Machines (SVM) are employed. The specific hyperparameters for each supervised learning model are described, considering model complexity and interpretability. Resampling strategies are used to compare the performance of various machine learning models and determine the best approach.

The study looks into whether clusters formed through unsupervised learning improve the prediction performance of the best machine learning model. This detailed analysis contributes to the knowledge of gene expression patterns and helps to construct predictive models for patient classification based on gene expression data.

Table of Contents

Abstract 2

Introduction 2

Preliminary Analysis: 3

Analysis 4

Discussions 12

Conclusion 12

Contribution 13

Ratings 14

References 13

Appendix 15

Word Count: 3,134

Introduction:

Gene expression data analysis plays a crucial role in unravelling biological mechanisms and identifying potential biomarkers for various disorders, notably cancer. In this project, our focus is to utilize both unsupervised and supervised machine learning algorithms to delve into a dataset comprising 2000 observed gene expression levels.

The gene expression dataset typically consists of measurements of gene expression levels across samples. Each row in the dataset represents a gene, and each column represents a sample (e.g., patient or experimental condition). The values in the dataset denote the expression levels of each gene in each sample, often represented as normalized intensities or counts. Our ultimate objective is to predict whether forthcoming patients are predisposed to invasive or non-invasive forms of cancer using supervised and unsupervised learning methods.

Preliminary Analysis:

Handling Missing Values:

The given sample dataset have 79 missing values. Among them, 76 were present in one row and the other row has 3 missing values. We have plotted box plots as shown in fig(1) to see the data distribution and it is observed that outliers are present in the data. Imputing values with the mean is not the correct way. So, the experiment was conducted to decide on which method to use to fill the missing values. Missing values imputed using k-NN Imputation method has given least misclassification error compared to values imputed using median method. The rows having missing values were filled with k-NN Imputation method to give the better results of the models.

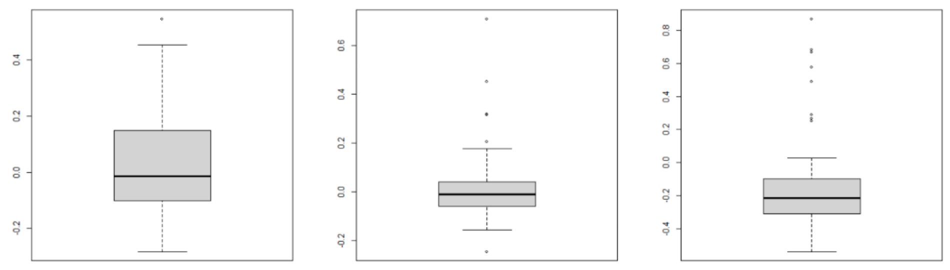


Fig.1

Analysis

We gathered the initial dataset from the provided csv file and we have set the seed value of 2312122 and we have considered top 2000 variables of the given data. We have replaced the class value having 1's with the value of zero's and 2's with the value of ones.

We have checked for the null values and we replaced the values using knn Imputation method. On the other hand we have replaced the null values using median.

Part 1

By consider supervised and unsupervised learning, dimensionality reduction of the 2000 observed gene expression data set is as follows:

Supervised Learning

Two-Sample T-test method: A t-test is a statistical analysis that compares data from two groups to determine the likelihood that the results obtained are significantly different from what is usually expected. We have used the Two-Sample T-test method, and the dataset has been reduced from 2000 to 314 columns. These maintained traits most likely record crucial information that helps distinguish between the classes. The decrease may lead to better classification performance and improved model interpretability.

Unsupervised Learning

Variance Method: A group of data points' dispersion or spread around their mean (average) is measured statistically via a concept called variance. The variance method aims to reduce a dataset's dimensions by selecting its most significant features that capture the most variance. The variance method reduced the dataset from 2000 to 56 columns having the most informative features and eliminated features with minimal variance.

R-tsne Method: t-SNE is a dimensionality reduction technique which points high-dimensional data to a lower-dimensional space using a measure like Euclidean distance to identify patterns between pairs of data. It reduces the data to two variables having the most specific features. We have used R-tsne method to perform QDA model as it requires very less number of variables for the analysis.

Part 2

We are examining groups of genes and groups of patients clustering through unsupervised learning techniques such as Principal Component Analysis (PCA), k-means clustering, and hierarchical clustering. The aim is to detect clusters/groups of genes and patients by analysing gene expression data. The process includes data preprocessing, dimensionality reduction via PCA, and clustering using k-means and hierarchical approaches.

We have performed all the techniques on both patients and genes.

In PCA analysis for patients, we use the dataset we got using the variance method. For groups of genes, we use the dataset we got from the two-sample t-test method. For others, like k clustering and hierarchical clustering for patients and genes, we use the dataset we got using the two-sample t-test method.

We are performing Principal Component Analysis (PCA) on gene/patient expression data, we summarised the result for groups of patients having 26 components showing cumulative proportion, which can explain the data variability to 90% as shown in fig(b) and for groups of genes having 37 components showing cumulative proportion which can explain the data variability to 90% as shown in fig(a), generating a bar plot to illustrate variance explained by each principal component. The analysis includes employing PCA to reduce data dimensionality and visualize gene clusters.

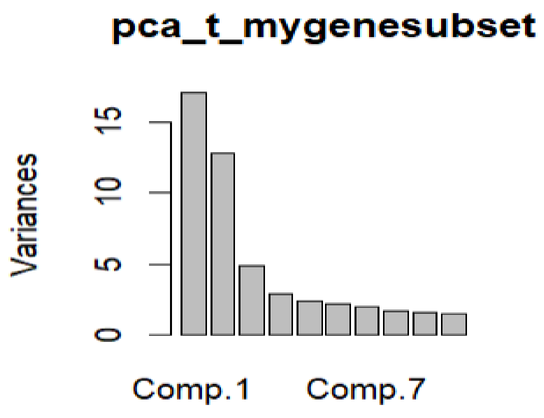


Fig.a

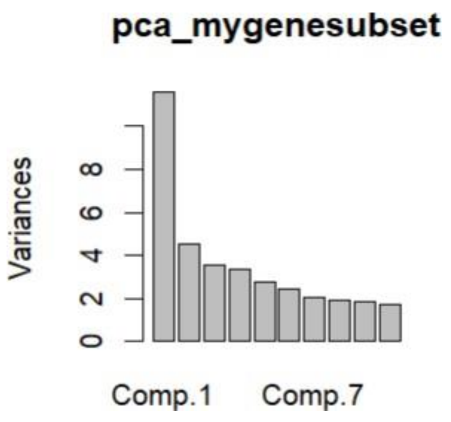
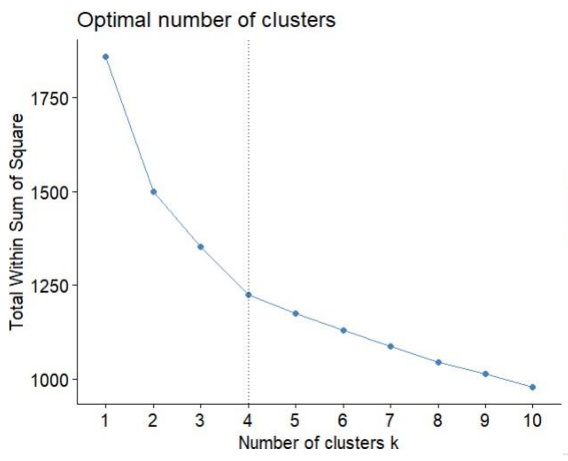


Fig.b

K-means clustering is a popular unsupervised machine learning algorithm used for partitioning a dataset into groups or clusters. This seeks to find the best arrangement of clusters based on the distribution of the data.

We are using the reduced data which we got the value from Two sample T-test method on to perform clustering on both, groups of genes and groups of patients. We scaled the data, Visualise the cluster and found the optimal value for the number of clusters to be used by using elbow method.

We got the value as 4 as shown in fig(c), which is the optimal value, and we plot the different clusters using the K-means function based on the data points. We used different colours to differentiate the clusters and plotted the graph shown in fig(d).



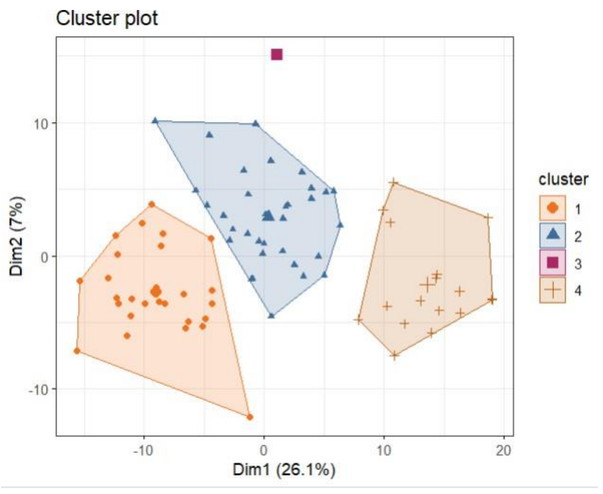


Fig.c **Fig.d**

Hierarchical clustering is a clustering method that constructs a hierarchy of clusters by treating each data point as a separate cluster and then progressively merging the closest pairs of clusters until all data points are linked into a single cluster or a specified stopping condition is met. This technique is valuable for identifying clusters within a dataset and revealing the relationships between data points in terms of their similarities or differences.

We are executing hierarchical clustering, a method for grouping similar data points into clusters based on their distance from each other. Specifically, the code calculates and illustrates the hierarchical clustering of scaled gene data by calculating the distance matrix using the Euclidean distance. The resulting dendrogram shown in fig(e) offers insights into the hierarchical organization of the data and how data points are grouped into clusters according to their similarities.

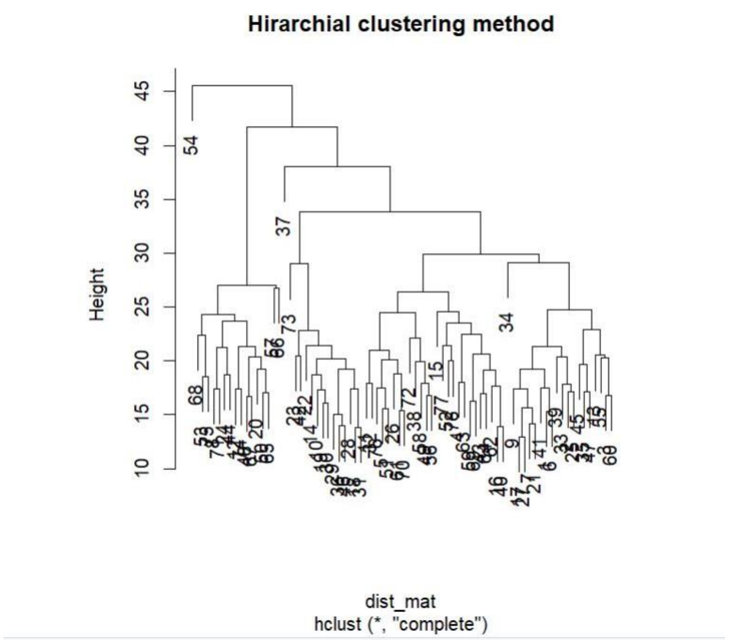


Fig.e

Part 3:

We filled the null values of the raw data using KNN Imputation and the median values, and we are using three sampling methods to predict the outcome variable.

1. We took the 2000 samples of data directly to predict the outcome variable through different models. We split the data into 70% train data and 30% test data and performed different models to predict the outcome variable.
2. We reduced the raw data using the t-test sampling method, by which we received 314 columns and 78 rows of data and data and performed different models to predict the outcome variable.
3. We used the re-sampling method for the t-test reduced data and performed different models to predict the outcome variable.

Logistic Regression:

Logistic regression is a statistical method to analyze data, predict the value of one variable based on another, and has limited outcomes. We are considering a binary logistic regression where there is a single binary dependent variable where there are two values labelled ‘0’ and ‘1’, and various independent variables which are having real values (continuous values).

We used t-test sampling data to train a logistic model and generate predictions. We converted the predictions to numeric data, assigned class labels, and constructed a confusion matrix. The misclassification error rate was 0.65, which was consistent for both KNN imputed and median data.

We used K-fold cross-validation with 5 re-sampling, dividing data into 4 training sets and 1 test set for validation. We stored misclassification errors and calculated their mean. KNN Imputed data had a mean error value of 0.3, while median data had a mean error value of 0.36.

Finally, we considered the data having 2000 columns and 78 rows and performed the model to calculate the misclassification. 0.61 is the value that we received for misclassification error in sampling and 0.44 for re-sampling and we are getting the same error for both KNN Imputed data as well as for median data.

Linear Discriminant Analysis (LDA):

LDA is a statistical technique that identifies a linear combination of features to separate classes of objects or events. It helps to find the most important factors that predict the outcome of interest. We used t-test sampling method to reduce the data. Then, we trained the LDA model with the train data, made predictions using the test data and created confusion matrix. The misclassification error was 0.42, which was the same for both KNN imputed data and median data.

We performed K-fold cross-validation with K=5 to train LDA models. The dataset was divided into four train sets and one test set for resampling. We stored misclassification errors and calculated the mean value. Mean error for KNN imputed data was 0.26, and for median data, it was 0.28.

Finally, we considered the data having 2000 columns and 78 rows and performed the model to calculate the misclassification. 0.5 is the value that we received for misclassification error in sampling and 0.33 for re-sampling and we are getting the same error for both KNN Imputed data as well as for median data.

Random Forest:

Random forest is a statistical method that clusters data into functional groups and calculates the probability of a data point belonging to a group based on the number of trees grown and variables used at each split. We used t-test sampling method on data and trained random forest model with train data. Predictions were made on test data which was converted to numeric data. We assigned class labels '0' and '1' and calculated 0.26 misclassification error for a tree of 3. Interestingly, we found the same error for KNN imputed and median data.

We used K-fold cross-validation with K=5 to resample the data and perform Random Forest. We calculated the mean misclassification errors, which were 0.29 for KNN Imputed data and 0.26 for median data, both for tree=50.

Finally, we considered the data having 2000 columns and 78 rows and performed the model to calculate the misclassification. 0.34 for tree = 8 is the value that we received for misclassification error in sampling and 0.36 for tree = 50 for re-sampling and we are getting the same error for both KNN Imputed data as well as for median data.

Support Vector Machine (SVM):

SVM is a supervised learning algorithm used for classification and regression tasks. Its main objective is to find the best line or decision boundary to separate data points belonging to different classes. We used t-test sampling to reduce the data, then trained an SVM model with the training data and used it to predict the test data. The predictions were converted to numeric data and assigned class labels. We created a confusion matrix to calculate the misclassification error, which was 0.42 for both KNN imputed data and median data.

We used a K-fold cross-validation with K=5 to perform SVM on the dataset. The mean misclassification error value was calculated and found to be 0.25, which was the same for both KNN imputed and median data.

Finally, we considered the data having 2000 columns and 78 rows and performed the model to calculate the misclassification. 0.57 is the value that we received for misclassification error in sampling and 0.33 for re-sampling and we are getting the same error for both KNN Imputed data as well as for median data.

Naïve Bayes:

Naive Bayes classifier is a machine learning model that calculates the probability of an input belonging to a particular class, assuming independence between features. We used t-test reduced data and Naïve Bayes model to make predictions on test data. Numeric functions were used to convert predictions into numeric data, and class labels ‘0’ and ‘1’ were assigned based on threshold values. The misclassification error calculated from the confusion matrix was 0.42, which was the same for both KNN imputed and median data.

We used K-fold cross-validation with K=5 to perform Naïve Bayes model, storing misclassification errors and considering the mean value. The error mean value is 0.21 for KNN Imputed data and 0.29 for median data.

Finally, we considered the data having 2000 columns and 78 rows and performed the model to calculate the misclassification. 0.42 is the value that we received for misclassification error in sampling and 0.35 for re-sampling and we are getting the same error for both KNN Imputed data as well as for median data.

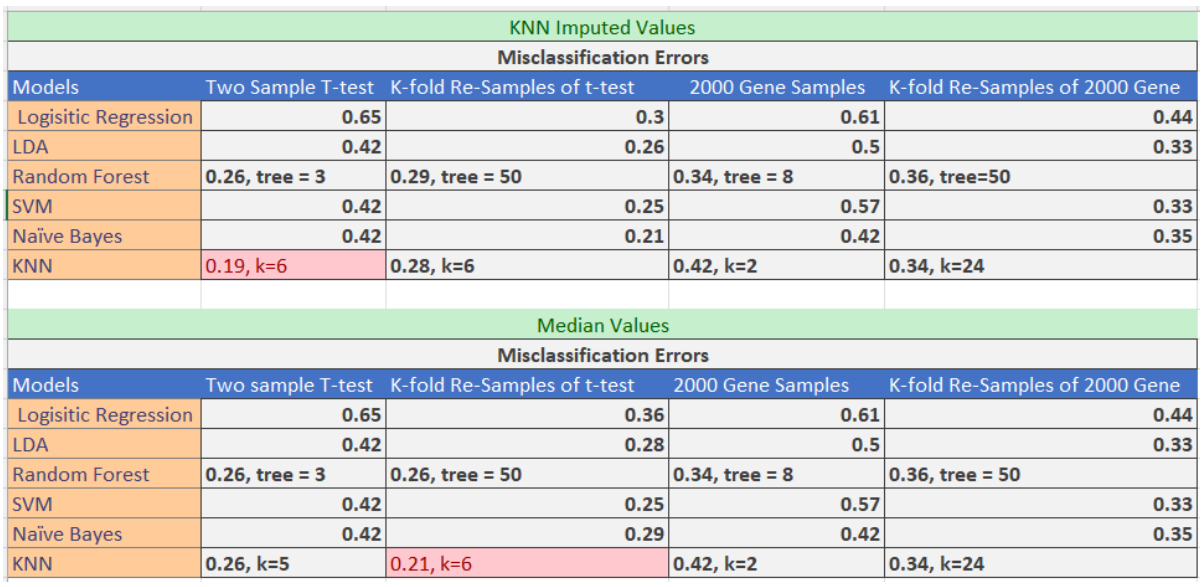
K-Nearest Neighbors (k-NN):

KNN is a non-parametric, supervised learning classifier that uses proximity to predict the grouping of a data point. Initially, we used the data which was reduced by t-test sampling method. We loaded the necessary library for performing the KNN model and trained the model using train data and we created confusion matrix using the numeric data and calculated the misclassification error which is 0.19 for k=6 in KNN imputed data and 0.26 for k=5 in median data.

We used K-fold cross-validation (K=5) to resample the dataset and perform KNN model. We performed the model and created the list for storing misclassification errors. The mean misclassification errors for KNN Imputed data were 0.28 (k=6) and for median data was 0.21 (k=6).

Finally, we considered the data having 2000 columns and 78 rows and performed the model to calculate the misclassification. 0.42 at k value 2 is the value that we received for misclassification error in sampling and 0.34 at k value 24 for re-sampling and we are getting the same error for both KNN Imputed data as well as for median data.

By considering the table 1, the misclassification error that we received is from KNN model in which 0.19 at k=6 for KNN Imputed data having two sample T-test and 0.21 at k=6 for median data having K-fold resampling of two sample T-test.



KNN Imputed Values				
Misclassification Errors				
Models	Two Sample T-test	K-fold Re-Samples of t-test	2000 Gene Samples	K-fold Re-Samples of 2000 Gene
Logisitic Regression	0.65	0.3	0.61	0.44
LDA	0.42	0.26	0.5	0.33
Random Forest	0.26, tree = 3	0.29, tree = 50	0.34, tree = 8	0.36, tree=50
SVM	0.42	0.25	0.57	0.33
Naïve Bayes	0.42	0.21	0.42	0.35
KNN	0.19, k=6	0.28, k=6	0.42, k=2	0.34, k=24

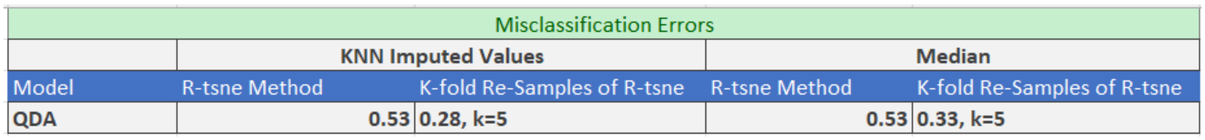
Median Values				
Misclassification Errors				
Models	Two sample T-test	K-fold Re-Samples of t-test	2000 Gene Samples	K-fold Re-Samples of 2000 Gene
Logisitic Regression	0.65	0.36	0.61	0.44
LDA	0.42	0.28	0.5	0.33
Random Forest	0.26, tree = 3	0.26, tree = 50	0.34, tree = 8	0.36, tree = 50
SVM	0.42	0.25	0.57	0.33
Naïve Bayes	0.42	0.29	0.42	0.35
KNN	0.26, k=5	0.21, k=6	0.42, k=2	0.34, k=24

Table 1

Quadratic Discriminant Analysis (QDA):

Quadratic Discriminant Analysis (QDA) assumes that each class follows a Gaussian distribution. It is a generative model, and the class-specific prior is simply the proportion of data points that belong to the class. We performed R-tsne dimensionality reduction method and we used R-tsne reduced data to operate the QDA model and make predictions on test data. The misclassification error calculated from the confusion matrix was 0.53, which was the same for both KNN imputed and median data.

We used K-fold cross-validation with K=5 to perform QDA model, storing misclassification errors and considering the mean value. The error mean value is 0.28 for KNN Imputed data and 0.33 for median data.



Misclassification Errors				
	KNN Imputed Values		Median	
Model	R-tsne Method	K-fold Re-Samples of R-tsne	R-tsne Method	K-fold Re-Samples of R-tsne
QDA	0.53	0.28, k=5	0.53	0.33, k=5

Table 2

Part 4:

Apart from all the models we used, we considered k-NN to be the best model from part 3, which gives the least misclassification error. Based on the results obtained from both K-means and hierarchical clustering of genes and patients, we used these clusters in the k-NN model. We observed that at k=7, the misclassification error was the same for both clusters, which was 0.22. By fitting the clusters for improvising the model, there is not much difference in the misclassification error compared to the k-NN model done in part 3.

Discussions

We have used the original data Initial dataset which contains 4949 variables. We have taken subset of 2000 variables and assigned 0 for all values of 1 and 1 for all vales of 2. Later we used k-NN Imputation and Median to fill the null values. We performed Two sample t-test which is supervised dimensionality reduction method which resulted having 314 variables.Also, we have used variance method which is unsupervised dimensionality reduction method which reduced the dataset to 56 columns. In addition to this, we used R-tsne method which is also unsupervised learning for only QDA analysis which reduced the data to only 2 columns, i.e. x1 and x2.

We have used unsupervised learning methods like PCA, k-means clustering, and hierarchical clustering to a gene expression dataset to identify clusters of genes and patients. To identify groups of patients we have used the dataset generated by variance method to perform PCA analysis while to identify groups of genes we have used the two-sample t-test generated dataset.In the next part, we have used supervised learning methods such as Logistic regression, random forest, k-NN, LDA, QDA, Naïve bayes, SVM models to find the misclassification errors. Compared to the sampling method resampling methods were giving better results by producing less misclassification errors as we were using k fold cross validation techniques. Among all the models, k-NN model gives the best result having least misclassification errors. k-NN imputed values performs better compared to the imputation of median values.

Conclusion

Considering all the results after the analysis of all the models, we have come to the conclusion that k-NN provides the best result having least misclassification error from the k-NN Imputed values. So, we fitted cluster to k-NN model to check whether we can improvise the model. But, in our case, there was not much difference in the misclassification error. Hence, we came to the conclusion that k-NN model gives the best outcome with or without fitting the clusters.

References

[1]. https://www.datacamp.com/tutorial/introduction-t-sne

[2]. https://www.researchgate.net/publication/228657549_Dimensionality_Reduction_A_Comparative_Review

[3]. J. C. Avendano, L. D. Otero and C. Otero, "Application of Statistical Machine Learning Algorithms for Classification of Bridge Deformation Data Sets," 2021 IEEE International Systems Conference (SysCon), Vancouver, BC, Canada, 2021.

[4]. J. Oyelade et al., "Data Clustering: Algorithms and Its Applications," 2019 19th International Conference on Computational Science and Its Applications (ICCSA), St. Petersburg, Russia, 2019.

[5]. K. G. Nisha and K. Sreekumar, "A review and analysis of machine learning and statistical approaches for prediction," 2017 International Conference on Inventive Communication and Computational Technologies (ICICCT), Coimbatore, India, 2017.

Contribution

Rishwanth Mithra (2311566) – Task 1, Task 3, Task 4, Group Report, Preliminary Analysis

Shradha Mishra (2312099) – Task 2, Task 4, Group Report, Presentation

Pragathi Girukala (2311894) – Task 2, Group Report

Raghunandan Kankanala (2312122) – Task 1, Group Report

Ratings

Rishwanth Mithra (2311566) – 5/5

Shradha Mishra (2312099) – 3.5/5

Pragathi Girukala (2311894) – 2.5/5

Raghunandan Kankanala (2312122) – 2/5

Appendix

```
InitialData <- read.csv(file="gene-expression-invasive-vs-noninvasive-cancer (1).csv")

str(InitialData)

dim(InitialData) # To determine the shape of the dataset


dimnames(InitialData)[[2]][4947:4949]

table(InitialData[4949])


set.seed(2312122)


team.gene.subset <- rank(runif(1:4948))[1:2000]

raw.gene.subset <- InitialData[team.gene.subset]


# Replace the class values of having 1 with 0's and having 2 with 1's

InitialData$Class[InitialData$Class == 1] <- 0

InitialData$Class[InitialData$Class == 2] <- 1


# Check null values

sum(is.na(raw.gene.subset))


# Find row number which contains missing values

rowSums(is.na(raw.gene.subset))


# Find out the names of the columns having missing values
```

```
mis_col <- names(which(colSums(is.na(raw.gene.subset))>0))

mis_col


# Plotting the box plot

par(mfrow = c(1,3))

boxplot(raw.gene.subset[,'AB029013'])

boxplot(raw.gene.subset[,'Contig53370_RC'])

boxplot(raw.gene.subset[,'NM_004774'])


library(DMwR2)

library(tidyr)


# Filling null values using KNN Imputation method and Median values


new_ds <- knnImputation(raw.gene.subset)


#new_ds<- new_ds %>%

 # mutate_all(~replace_na(.,median(.,na.rm = TRUE)))

sum(is.na(new_ds))


# Part 1

# Dimensional Reduction Methods
# Supervised Learning

# To perform Two Sample T-test Method

library(dplyr)
t_test_results <- new_ds %>%

  #select(-gene_id) %>%

summarise_all(~t.test(. ~ InitialData$Class)$p.value)

t <- t_test_results[,t_test_results < 0.05]

sig.genes <- colnames(t)

reduced.data <- new_ds[sig.genes]
reduced.data$Class <- InitialData$Class    # Initializing class to the datasets
new_ds$Class <- InitialData$Class
ncol(reduced.data)
nrow(reduced.data)

# Unsupervised Learning

# Variance Method
```

```
var_data <- apply(new_ds, 2, var)

var_data_col <- names(var_data[var_data > 0.25])

var_data_subset <- new_ds[, var_data_col]

ncol(var_data_subset)

nrow(var_data_subset)


# R-tsne Method


install.packages('Rtsne')

library(Rtsne)

tsne <- Rtsne(new_ds, perplexity=10)

tsne_data<-data.frame(tsne[['Y']])


# Part 2

# Groups of Patients


# PCA

library(stats)

pca_mygenesubset <- princomp(var_data_subset, cor = TRUE) # Performing PCA using
variance method dataset


summary(pca_mygenesubset)


pca_comp <- pca_mygenesubset$scores[,1:26]
```

```
ncol(pca_comp)


# plotting the bar graph

plot(pca_mygenesubset)




# K-Means Clustering


library(ggpubr)

library(factoextra)


#Scaling the patients data


reduced.data$Class <- NULL

scaled_data <- scale(reduced.data)




#cluster visualization

cluster_result <- mygene_clust$cluster


#Elbow method

fviz_nbclust(reduced.data, # dataset

             kmeans, # clustering algorithm

             nstart = 25,
            iter.max = 200, # the number of iterations allowed

            method = "wss")+geom_vline(xintercept = 4, linetype = 3 )


#plotting the cluster


mygene_clust <- kmeans(reduced.data, centers = 4, nstart = 20)

table(mygene_clust$cluster)


fviz_cluster(mygene_clust, data = reduced.data,

            palette = c("#EF731E","#3F6E9A","#AD2765", '#BC6213'),

            geom = "point",

            ellipse.type = "convex",

            ggtheme = theme_bw()

)


table(mygene_clust$cluster)



# Hierarchical clustering


dist_mat <- dist(scaled_data, method = "euclidean")

hclust_result <- hclust(dist_mat, method = "complete")


#visualizing hierarchial clustering
```

```
plot(hclust_result, main = "Hirarchial clustering method")


clusters_ <- cutree(hclust_result, k=2)

table(clusters_)


# Groups of Genes

transpose_t_data_subset <- t(reduced.data)

table(transpose_t_data_subset)


# PCA

library(stats)

pca_t_mygenesubset <- princomp(transpose_t_data_subset, cor = TRUE)  # Performing PCA
using variance method Transposed dataset


summary(pca_t_mygenesubset)


pca_t_comp <- pca_t_mygenesubset$scores[,1:37]

table(pca_t_comp)


#plotting the bar graph


plot(pca_t_mygenesubset)




# K-Means Clustering
```

```
library(ggpubr)

library(factoextra)


#Scaling the gene data


scaled_t_data <- scale(reduced.data)


mygene_t_clust <- kmeans(reduced.data, centers = 2, nstart = 50)


#visualizing the cluster

cluster_t_result <- mygene_t_clust$cluster


#Elbow method

fviz_nbclust(reduced.data, # data

             kmeans, # clustering algorithm

             nstart = 25,

             iter.max = 200, # the number of iterations allowed

             method = "wss")+ geom_vline(xintercept = 4, linetype = 3 )


#plotting the cluster


mygene_t_clust <- kmeans(reduced.data, centers = 4, nstart = 20)

table(mygene_t_clust$cluster)
```

```
fviz_cluster(mygene_t_clust, data = reduced.data,
        palette = c("#EF731E","#3F6E9A","#AD2765", '#BC6213'),
        geom = "point",
        ellipse.type = "convex",
        ggtheme = theme_bw()
)

table(mygene_t_clust$cluster)

# Hierarchial clustering

dist_t_mat <- dist(scaled_t_data, method = "euclidean")
hclust_t_result <- hclust(dist_t_mat, method = "complete")

#visualizing hierarchical clustering
plot(hclust_t_result, main = "Hierarchical clustering method")

clusters_t <- cutree(hclust_t_result, k=2)
table(clusters_t)

# Part 3

##################################### Sampling using Two sample t-test data
###############################################
```

```
# Logistic Regression

#train_data$Class<-new_ds$Class
library(caTools)

# Splitting the data to train and test
split <- sample.split(reduced.data, SplitRatio = 0.7, set.seed(2312122))
train_data <- subset(reduced.data, split == TRUE)
test_data <- subset(reduced.data, split == FALSE)

# Checking the null values
sum(is.na(train_data))
train_data<- na.omit(train_data)
test_data<- na.omit(test_data)
table(train_data$Class)
table(test_data$Class)

# Performing the Logistic regression model
log_model <- glm(Class ~ ., data = train_data, family = "binomial")
summary(log_model)

# Predicting the model
log_model.predict <- predict(log_model,newdata = test_data, type = "response")
```

```
summary(log_model.predict)

str(log_model.predict)


# Converting the predicted values to numeric values

log_numeric_data <- as.numeric(log_model.predict)

log_numeric_data

log_numeric_data <- na.omit(log_numeric_data)




predicted_class_log <- ifelse(log_numeric_data > 0.5,1,0)

predicted_class_log

test_data <- na.omit(test_data)

test_data


# Constructing confusion matrix

log_conf_matrix <- table(list(predicted_class_log, test_data$Class))

log_conf_matrix


# Calculating misclassification error

log_misclassification_error <- 1 - sum(diag(conf_matrix)) / sum(conf_matrix)

log_misclassification_error


# LDA
```

```
library(MASS)


# Performing the LDA model

lda_model_1 <- lda(Class ~ ., data = train_data)

summary(lda_model_1)


# Predicting the model

model_lda.predict <- predict(lda_model_1,newdata = test_data)$class

summary(model_lda.predict)


# Constructing confusion matrix

conf_matrix_lda <- table(model_lda.predict, test_data$Class)

conf_matrix_lda


# Calculating misclassification error

misclassification_error_lda <- 1 - sum(diag(conf_matrix_lda)) / sum(conf_matrix_lda)

misclassification_error_lda




# Random Forest

train_data <- na.omit(train_data)


library(randomForest)

e_rf = list() # Creating the list to store the errors
```

```
for (i in 1:15){

  # Performing the Random Forest model
  rf_model_1 <- randomForest(Class ~ ., data = train_data, ntree = i,set.seed(2312122))

  # Predicting the model
  predict.rf<- predict(rf_model_1, newdata = test_data, type = "response")

  str(predict.rf)

  # Converting the predicted values to numeric values
  numeric_rf <- as.numeric(predict.rf)
  predicted_class_rf <- ifelse(numeric_rf > 0.5, 1, 0)

  # Constructing confusion matrix
  rf_conf_matrix <- table(list(predicted_class_rf, test_data$Class))
  rf_conf_matrix

  # Calculating misclassification error
  rf_misclassification_error <- 1 - sum(diag(rf_conf_matrix)) / sum(rf_conf_matrix)
  rf_misclassification_error
  e_rf <- append(e_rf,rf_misclassification_error)
}
```

```
e_rf


# SVM
library(caret)
library(e1071)

# Performing the SVM model
svm_mod <- svm(Class ~ ., data = train_data, kernel = "radial")

# Predicting the model
predict_svm <- predict(svm_mod, newdata = test_data)

# Summarize the predictions
summary(predict_svm)

# Converting predictions to numeric data
numeric_svm <- as.numeric(predict_svm)

# Assign class labels based on a threshold
svm_predicted_class <- ifelse(numeric_svm > 0.5, 1, 0)

conf_matrix_svm <- confusionMatrix(as.factor(svm_predicted_class),
as.factor(test_data$Class))
```

```
# Constructing confusion matrix
conf_matrix_svm <- table(svm_predicted_class, test_data$Class)

conf_matrix_svm

# Calculating misclassification error
misclassification_svm <- 1 - sum(diag(conf_matrix_svm)) / sum(conf_matrix_svm)
misclassification_svm

# Naive Bayes

# Load the necessary library
library(e1071)

# performing the Naive Bayes model
model_nvb <- naiveBayes(Class ~ ., data = train_data)

# Make predictions on the model
predict_nvb <- predict(model_nvb, newdata = test_data)

# Summarize the predictions
summary(predict_nvb)

# Converting predicted values to numeric data
```

```
numeric_nvb <- as.numeric(predict_nvb)


# Assign class labels based on a threshold value

predicted_class_nvb <- ifelse(numeric_nvb > 0.5, 1, 0)


# generate confusion matrix

nvb_conf_matrix <- table(predicted_class_nvb, test_data$Class)


nvb_conf_matrix


# Calculate misclassification error

nvb_error <- 1 - sum(diag(nvb_conf_matrix)) / sum(nvb_conf_matrix)

nvb_error




# KNN


library(class)

test_data = na.omit(test_data)

e_knn <- list() # Creating list to store error values

for (i in 1:10){


  # performing the KNN model
  knn_m<- knn(train = train_data, test = test_data, cl=train_data$Class, k = i)
  summary(knn_model)

  # generate confusion matrix
  matrix_knn <- table(knn_m, test_data$Class)
  matrix_knn

  #Calculate misclassification error
  error_knn <- 1 - sum(diag(matrix_knn)) / sum(matrix_knn)
  error_knn
  e_knn <- append(e_knn,error_knn)
}
e_knn
sum(is.na(train_data))
#na.omit(train_data)

# QDA

# Initializing Class Variable
tsne_data$Class<-new_ds$Class

# Split data to train and test
split_ts <- sample.split(tsne_data, SplitRatio = 0.7, set.seed(2312122))
train_data_ts <- subset(tsne_data, split == TRUE)
```

```
test_data_ts <- subset(tsne_data, split == FALSE)

sum(is.na(train_data_ts))

train_data_ts<- na.omit(train_data_ts)

test_data_ts<- na.omit(test_data_ts)



library(MASS)


# performing the Naive Bayes model

qda_model_1 <- qda(Class ~ ., data = train_data_ts)

summary(qda_model_1)


# Make predictions on the model

model_qda.predict <- predict(qda_model_1,newdata = test_data_ts)$class

summary(model_qda.predict)


# generate confusion matrix

conf_matrix_qda <- table(model_qda.predict, test_data_ts$Class)

conf_matrix_qda


#Calculate misclassification error

misclassification_error_qda <- 1 - sum(diag(conf_matrix_qda)) / sum(conf_matrix_qda)

misclassification_error_qda
```

```
############################################ Re-sampling K-Fold Technique
#################################



# Logistic Regression


library(caret)


k <- 5

error <- list()


for (i in 1:(k-1)) {

  set.seed(2312122)

  indices <- sample(1:nrow(train_data), replace = FALSE)

  split <- cut(indices, breaks = k, labels = FALSE)


  # Perform logistic regression model

  logistic_m_2 <- glm(Class ~ ., data = train_data[split != i, ], family = "binomial")


  # Prediction of the validation set

  predicted_log_2 <- predict(logistic_m_2, newdata = train_data[split == i, ], type =
"response")


  # Convert predicted probabilities to class labels

  predicted_class_log_2 <- ifelse(predicted_log_2 > 0.5, 1, 0)
  # Create confusion matrix
  conf_matrix_log_2 <- confusionMatrix(as.factor(predicted_class_log_2),
as.factor(train_data$Class[split == i]))

  # Calculate misclassification error
  log_Mis.error <- 1 - conf_matrix_log_2$overall["Accuracy"]

  error <- append(error, log_Mis.error)
}

error
e <- unlist(error)

# mean error
mean(e)

# KNN

library(caret)

# Initialize minimum error to 1
min_error <- 1
error <- list()
k_fold <- 5
```

```
# Sample indices
set.seed(2312122)
indices <- sample(1:nrow(train_data), replace = FALSE)
for (j in 1:50){
  for (i in 1:(k_fold-1)) {
    split <- cut(indices, breaks = k, labels = FALSE)

  # Perform KNN model
    knn_model_2 <- knn(train = train_data[split != i, -ncol(train_data)],
                    test = train_data[split == i, -ncol(train_data)],
                    cl = train_data$Class[split != i],
                    k = j)  # Adjust k as needed

  # Prediction of the model
    predicted_knn_2 <- knn_model_2

  # Create confusion matrix
    conf_matrix_knn_2 <- confusionMatrix(as.factor(predicted_knn_2),
                                         as.factor(train_data$Class[split == i]))

  # Calculate misclassification error
    knn_error2 <- 1 - conf_matrix_knn_2$overall["Accuracy"]

    error <- append(error, knn_error2)
  }


  error
  e <- unlist(error)
  mean_e<-mean(e)
  mean_e
  # Store minimum error in a variable
  if (mean_e < min_error){
  min_error<-mean_e
    v <- j
  }
}
min_error
v


# LDA


library(caret)


k <- 5
error <- list()


# Sampling the indices
set.seed(2312122)
```

```
indices <- sample(1:nrow(train_data), replace = FALSE)

for (i in 1:(k-1)) {
  split <- cut(indices, breaks = k, labels = FALSE)

  # Perform LDA model
  lda_model_2 <- lda(Class ~ ., data = train_data[split != i, ])

  # Predicting the model
  predicted_lda_2 <- predict(lda_model_2, newdata = train_data[split == i, ])

  # Convert predicted class to labels
  class_lda_2 <- predicted_lda_2$class

  # Generating confusion matrix
  conf_matrix_lda_2 <- confusionMatrix(as.factor(class_lda_2),
                                       as.factor(train_data$Class[split == i]))

  # Calculate misclassification error
  lda_error_2 <- 1 - conf_matrix_lda_2$overall["Accuracy"]

  error <- append(error, lda_error_2)
}
```

```
# Print errors
error
e <- unlist(error)

# Calculate mean error
mean(e)

#SVM

library(caret)
library(e1071)

k <- 5
error <- list()

for (i in 1:(k-1)) {
  set.seed(2312122)
  indices <- sample(1:nrow(train_data), replace = FALSE)

  split <- cut(indices, breaks = k, labels = FALSE)

  # Train SVM model
  model_2_svm <- svm(Class ~ ., data = train_data[split != i, ], kernel = "radial")
  # Predicting the validation set
  svm_predicted_2 <- predict(model_2_svm, newdata = train_data[split == i, ])


  predicted_class_svm_2 <- ifelse(svm_predicted_2 > 0.5, 1, 0)


  # generate confusion matrix
  conf_matrix_svm_2 <- confusionMatrix(as.factor(predicted_class_svm_2),
                                       as.factor(train_data$Class[split == i]))


  # Calculate misclassification error
  svm_2_error <- 1 - conf_matrix_svm_2$overall["Accuracy"]


  # Append error to list
  error <- append(error, svm_2_error)
}


error
e <- unlist(error)


# Calculating mean error
mean(e)


# Random Forest
```

```
library(caret)


min_error <- 1

error <- list()

k_fold <- 5

# Sampling the indices

set.seed(2312122)

indices <- sample(1:nrow(train_data), replace = FALSE)

for (j in 1:50) {

  for (i in 1:(k_fold-1)) {

    split <- cut(indices, breaks = k_fold, labels = FALSE)


    # Performing Random Forest model

    model_rf_2 <- randomForest(Class ~ ., data = train_data[split != i, ], ntree = j)  # Number
of trees in the forest

    # Adjust parameters as needed


    # Predicting the data

    predicted_2_rf <- predict(model_rf_2, newdata = train_data[split == i,],type ="response")

    predicted_class_2_rf <- ifelse(predicted_2_rf > 0.5, 1, 0)


    # Creating confusion matrix

    rf_2_conf_matrix <- confusionMatrix(as.factor(predicted_class_2_rf),

                                         as.factor(train_data$Class[split == i]))
    # Calculating the misclassification error
    rf_2_error <- 1 - rf_2_conf_matrix$overall["Accuracy"]

    # Appending error to list
    error <- append(error, rf_2_error)
  }


  e <- unlist(error)
  mean_e <- mean(e)
  # Storing the minimum value
  if (mean_e < min_error) {
  min_error <- mean_e
    var <- j
  }
}

min_error
var


# Naive Bayes

library(caret)
library(e1071)


k <- 5

error <- list()


for (i in 1:(k-1)) {

  set.seed(2312122)

  indices <- sample(1:nrow(train_data), replace = FALSE)

  split <- cut(indices, breaks = k, labels = FALSE)


  # Train Naive Bayes model

  model_2_nb <- naiveBayes(Class ~ ., data = train_data[split != i, ])


  # Predicting the validation set

  nb_2_predicted <- predict(model_2_nb, newdata = train_data[split == i, ])


  # Create the confusion matrix

  conf_matrix_nb_2 <- confusionMatrix(as.factor(nb_2_predicted),

                                      as.factor(train_data$Class[split == i]))


  # Calculate the misclassification error

  nb_error_2 <- 1 - conf_matrix_nb_2$overall["Accuracy"]

  error <- append(error, nb_error_2)

}
```

```
error

e <- unlist(error)


# Calculate mean error

mean(e)




# QDA


library(caret)


k <- 5

error <- list()


# Sample indices once before the loop

set.seed(2312122)

indices <- sample(1:nrow(train_data_ts), replace = FALSE)


for (i in 1:(k-1)) {

  split_ts <- cut(indices, breaks = k, labels = FALSE)


  # Train QDA model

  qda_model_2 <- qda(Class ~ ., data = train_data_ts[split_ts != i, ])
  # Predict on the validation set
  predicted_qda_2 <- predict(qda_model_2, newdata = train_data_ts[split_ts == i, ])

  # Convert predicted class to labels
  class_qda_2 <- predicted_qda_2$class

  # Create confusion matrix
  conf_matrix_qda_2 <- confusionMatrix(as.factor(class_qda_2),
                                       as.factor(train_data_ts$Class[split_ts == i]))

  # Calculate misclassification error
  qda_error_2 <- 1 - conf_matrix_qda_2$overall["Accuracy"]

  # Append error to list
  error <- append(error, qda_error_2)
}

# Print errors
error
e <- unlist(error)

# Calculate mean error
mean(e)
```

```
  ############################################# 2000 Gene Data
#########################################################

# Logistic Regression

library(caTools)

# Split the data into train and test
split <- sample.split(new_ds, SplitRatio = 0.7, set.seed(2312122))
train_data_2000 <- subset(new_ds, split == TRUE)
test_data_2000 <- subset(new_ds, split == FALSE)
#train_data_2000$Class <- InitialData$Class

table(train_data_2000$Class)
table(test_data_2000$Class)

# Perform the Logistic Regression
model_log_3 <- glm(Class ~ ., data = train_data_2000, family = "binomial")
summary(model_log_3)

# Predicting the model
model_3_predict_log <- predict(model_log_3,newdata = test_data_2000, type = "response")
summary(model_3_predict_log)
str(model_3_predict_log)
```

```
# Converting the predicted values to numeric values

numeric_data_log<- as.numeric(model_3_predict_log)

numeric_data_log

numeric_data_log <- na.omit(numeric_data)

predicted_class_log_3 <- ifelse(numeric_data_log > 0.5,1,0)

predicted_class_log_3

test_data_2000 <- na.omit(test_data_2000)

test_data_2000


# Generating the confusion Matrix

conf_matrix_log_3 <- table(list(predicted_class_log_3, test_data_2000$Class))

conf_matrix_log_3


# Calculating the error

misclassification_error_log_3 <- 1 - sum(diag(conf_matrix_log_3)) /
sum(conf_matrix_log_3)

misclassification_error_log_3


# LDA

#install.packages('MASS')

library(MASS)


# Perform the LDA model

model_lda_3 <- lda(Class ~ ., data = train_data_2000)
```

```
summary(model_lda_3)


# Predicting the model

model_3_predict_lda <- predict(model_lda_3,newdata = test_data_2000)$class

summary(model_3_predict_lda)

model_3_predict_lda

test_data_2000$Class


# Creating the confusion Matrix

conf_matrix_3_lda <- table(model_3_predict_lda, test_data_2000$Class)

conf_matrix_3_lda


# Calculating the misclassification error

error_3_lda <- 1 - sum(diag(conf_matrix_3_lda)) / sum(conf_matrix_3_lda)

error_3_lda


# Random Forest

train_data_2000 <- na.omit(train_data_2000)

library(randomForest)

e_rf = list() # Creating a list to store errors


for (i in 1:100){


  # Performing the random forest
  rf_3_model <- randomForest(Class ~ ., data = train_data_2000, ntree = i,set.seed(2312122))


  # Predicting the model
  model_3_rf<- predict(rf_3_model, newdata = test_data_2000, type = "response")  #
Corrected: rf_model instead of model.predict


  str(model.predict.rf)


  # Converting the predicted values to numeric
  numeric_data_3_rf <- as.numeric(model_3_rf)
  predicted_class_3_rf <- ifelse(numeric_data_3_rf > 0.5, 1, 0)


  # Generating the confusion matrix
  conf_matrix_3_rf <- table(list(predicted_class_3_rf, test_data_2000$Class))
  conf_matrix_3_rf


  # Calculating the misclassification error
  error_3_rf <- 1 - sum(diag(conf_matrix_3_rf)) / sum(conf_matrix_3_rf)
  error_3_rf
  e_rf <- append(e_rf,error_3_rf)
}
e_rf


# SVM
```

```
library(e1071)


# Perform the SVM model

svm_model_3 <- svm(Class ~ ., data = train_data_2000, kernel = "radial")


# predicting the test data

predict_svm_3 <- predict(svm_model_3, newdata = test_data_2000)

summary(predict_svm_3)


# Convert predicted values to numeric data

numeric_data_svm_3 <- as.numeric(predict_svm_3)

predicted_class_svm_3 <- ifelse(numeric_data_svm_3 > 0.5, 1, 0)


# Creating the confusion matrix

conf_matrix_3_svm <- confusionMatrix(as.factor(predicted_class_svm_3),
as.factor(test_data_2000$Class))

conf_matrix_3_svm <- table(predicted_class_svm_3, test_data_2000$Class)


conf_matrix_3_svm


# Calculating the misclassification error

error_svm_3 <- 1 - sum(diag(conf_matrix_3_svm)) / sum(conf_matrix_3_svm)

error_svm_3


# Naive Bayes
```

```
library(e1071)


# Performing the Naive Bayes model

model_3_nb <- naiveBayes(Class ~ ., data = train_data_2000)


# Predictions on validation set

model_pred_nb_3 <- predict(model_3_nb, newdata = test_data_2000)

summary(model_pred_nb_3)


# Convert predicted values to numeric data

numeric_nb_3 <- as.numeric(model_pred_nb_3)


# Assign class labels based on a threshold

class_nb_3 <- ifelse(numeric_nb_3 > 0.5, 1, 0)


# Creating the confusion matrix

conf_matrix_nb_3 <- table(class_nb_3, test_data_2000$Class)

conf_matrix_nb_3


# Calculating the misclassification error

error_nb_3 <- 1 - sum(diag(conf_matrix_nb_3)) / sum(conf_matrix_nb_3)

error_nb_3
```

```
# KNN


library(class)

train_data_2000 = na.omit(train_data_2000)

test_data_2000 = na.omit(test_data_2000)

e_knn_3 <- list() # Creating the list

for (j in (1:10)){

  # Performing the KNN Model

  knn_3_model <- knn(train = train_data_2000, test = test_data_2000, cl=
train_data_2000$Class, k=j)


  # Creating the matrix

  conf_matrix_3_knn <- table(list(knn_3_model, test_data_2000$Class))

  conf_matrix_3_knn


  # Calculating the error

  error_3_knn <- 1 - sum(diag(conf_matrix_3_knn)) / sum(conf_matrix_3_knn)

  error_3_knn

  e_knn_3 <- append(e_knn_3,error_3_knn)

}

e_knn_3
```

```
################################### Re-Sampling of 2000 Gene Data
##################################################

# Logistic Regression

library(caret)

k <- 5
error <- list()

for (i in 1:(k-1)) {
  set.seed(2312122)
  indices <- sample(1:nrow(train_data_2000), replace = FALSE)
  split <- cut(indices, breaks = k, labels = FALSE)

  # Performing the logistic regression model
  logistic_m_2 <- glm(Class ~ ., data = train_data_2000[split != i, ], family = "binomial")

  # Predicting the model
  predicted_log_2 <- predict(logistic_m_2, newdata = train_data_2000[split == i, ], type =
"response")
  predicted_class_log_2 <- ifelse(predicted_log_2 > 0.5, 1, 0)

  # Creating the confusion matrix
  conf_matrix_log_2 <- confusionMatrix(as.factor(predicted_class_log_2),
as.factor(train_data_2000$Class[split == i]))


  # Calculating the misclassification error

  log_Mis.error <- 1 - conf_matrix_log_2$overall["Accuracy"]

  error <- append(error, log_Mis.error)

}


error

e <- unlist(error)


# Calculate mean error

mean(e)


# KNN


library(caret)




min_error <- 1

error <- list()

k_fold <- 5

# Sampling the indices

set.seed(2312122)

sum(is.na(train_data_2000))
```

```
indices <- sample(1:nrow(train_data_2000), replace = FALSE)
for (j in 1:50){
  for (i in 1:(k_fold-1)) {
    split <- cut(indices, breaks = k, labels = FALSE)

    # Perform KNN model
    knn_model_2 <- knn(train = train_data_2000[split != i, -ncol(train_data_2000)],
                       test = train_data_2000[split == i, -ncol(train_data_2000)],
                       cl = train_data_2000$Class[split != i],
                       k = j)  # Adjust k value

    # Predict the model
    predicted_knn_2 <- knn_model_2

    # Creating the confusion matrix
    conf_matrix_knn_2 <- confusionMatrix(as.factor(predicted_knn_2),
                                         as.factor(train_data_2000$Class[split == i]))

    # Calculating the misclassification error
    knn_error2 <- 1 - conf_matrix_knn_2$overall["Accuracy"]
    error <- append(error, knn_error2)
  }

  error
  e <- unlist(error)
  mean_e<-mean(e)
  mean_e  # Store minimum error in a variable
  if (mean_e < min_error){
    min_error<-mean_e
    v <- j
  }
}
min_error
v

# LDA

library(caret)

k <- 5
error <- list()

# Sample the indices
set.seed(2312122)
indices <- sample(1:nrow(train_data_2000), replace = FALSE)

for (i in 1:(k-1)) {
  split <- cut(indices, breaks = k, labels = FALSE)
  # Training the LDA model
  lda_model_2 <- lda(Class ~ ., data = train_data_2000[split != i, ])

  # Predicting the validation set
  predicted_lda_2 <- predict(lda_model_2, newdata = train_data_2000[split == i, ])

  class_lda_2 <- predicted_lda_2$class

  # Create the confusion matrix
  conf_matrix_lda_2 <- confusionMatrix(as.factor(class_lda_2),
                                       as.factor(train_data_2000$Class[split == i]))

  # Calculate the misclassification error
  lda_error_2 <- 1 - conf_matrix_lda_2$overall["Accuracy"]
  error <- append(error, lda_error_2)
}

# Print errors
error
e <- unlist(error)

# Calculate mean error
mean(e)
```

```
#SVM
library(caret)
library(e1071)

k <- 5
error <- list()

for (i in 1:(k-1)) {
  set.seed(2312122) # Sampling the indices
  indices <- sample(1:nrow(train_data_2000), replace = FALSE)

  split <- cut(indices, breaks = k, labels = FALSE)

  # Training the SVM model
  model_2_svm <- svm(Class ~ ., data = train_data_2000[split != i, ], kernel = "radial")

  # Predicting the validation set
  svm_predicted_2 <- predict(model_2_svm, newdata = train_data_2000[split == i, ])

  predicted_class_svm_2 <- ifelse(svm_predicted_2 > 0.5, 1, 0)

  # Creating the confusion matrix
  conf_matrix_svm_2 <- confusionMatrix(as.factor(predicted_class_svm_2),
                                    as.factor(train_data_2000$Class[split == i]))

  # Calculate misclassification error
  svm_2_error <- 1 - conf_matrix_svm_2$overall["Accuracy"]
  error <- append(error, svm_2_error)
}

error
e <- unlist(error)

# Calculate mean error
mean(e)

# Random Forest

library(caret)

min_error <- 1
error <- list()
k_fold <- 5
# Sample the indices values
set.seed(2312122)
indices <- sample(1:nrow(train_data_2000), replace = FALSE)
for (j in 1:50) {
  for (i in 1:(k_fold-1)) {
    split <- cut(indices, breaks = k_fold, labels = FALSE)

    # Performing the Random Forest model
    model_rf_2 <- randomForest(Class ~ ., data = train_data_2000[split != i, ], ntree = j)

    # Predicting the validation set
    predicted_2_rf <- predict(model_rf_2, newdata = train_data_2000[split == i,],type
="response")
    predicted_class_2_rf <- ifelse(predicted_2_rf > 0.5, 1, 0)

    # Creating the confusion matrix
    rf_2_conf_matrix <- confusionMatrix(as.factor(predicted_class_2_rf),
                                        as.factor(train_data_2000$Class[split == i]))

    # Calculating the misclassification error
    rf_2_error <- 1 - rf_2_conf_matrix$overall["Accuracy"]

    error <- append(error, rf_2_error)
  }


  e <- unlist(error)
  mean_e <- mean(e) # Store the minimum error in a variable
  if (mean_e < min_error) {
    min_error <- mean_e
    var <- j
  }
}

min_error
var


# Naive Bayes

library(caret)
library(e1071)

k <- 5
error <- list()

for (i in 1:(k-1)) {
  set.seed(2312122) # Sampling the indices
  indices <- sample(1:nrow(train_data_2000), replace = FALSE)
  split <- cut(indices, breaks = k, labels = FALSE)

  # Training the Naive Bayes model
  model_2_nb <- naiveBayes(Class ~ ., data = train_data_2000[split != i, ])
  # Predicting the trained model
  nb_2_predicted <- predict(model_2_nb, newdata = train_data_2000[split == i, ])


  # Creating the confusion matrix
  conf_matrix_nb_2 <- confusionMatrix(as.factor(nb_2_predicted),
                                      as.factor(train_data_2000$Class[split == i]))


  # Calculating the misclassification error
  nb_error_2 <- 1 - conf_matrix_nb_2$overall["Accuracy"]


  error <- append(error, nb_error_2)
}


# Print errors
error
e <- unlist(error)
mean(e)


# Part 4


reduced.data$Class <- new_ds$Class
reduced.data$Cluster<- mygene_clust$cluster
```

```
# KNN Model


library(caret)


min_error <- 1
error <- list()
k_fold <- 5
# Sample the indices
set.seed(2312122)
indices <- sample(1:nrow(reduced.data), replace = FALSE)
for (j in 1:50){
  for (i in 1:(k_fold-1)) {
    split <- cut(indices, breaks = k, labels = FALSE)


    # Training the KNN model
    knn_model_2 <- knn(train = reduced.data[split != i, -ncol(reduced.data)],
                       test = reduced.data[split == i, -ncol(reduced.data)],
                       cl = reduced.data$Class[split != i],
                       k = j)  # Adjust k the value


    # Predicting the validation set
    predicted_knn_2 <- knn_model_2


    # Creating the confusion matrix
    conf_matrix_knn_2 <- confusionMatrix(as.factor(predicted_knn_2),
                                         as.factor(reduced.data$Class[split == i]))

    # Calculating the misclassification error
    knn_error2 <- 1 - conf_matrix_knn_2$overall["Accuracy"]

    error <- append(error, knn_error2)
  }

  error
  e <- unlist(error)
  mean_e<-mean(e)
  mean_e    # Storing the minimum error to a variable
  if (mean_e < min_error){
    min_error<-mean_e
    v <- j
  }
}
min_error
v

# Hierarchical Clustering

reduced.data$Cluster<- NULL
reduced.data$clusters_ <- clusters_


# KNN Model


library(caret)


min_error <- 1
error <- list()
k_fold <- 5
# Sample the indices
set.seed(2312122)
indices <- sample(1:nrow(reduced.data), replace = FALSE)
for (j in 1:50){
  for (i in 1:(k_fold-1)) {
    split <- cut(indices, breaks = k, labels = FALSE)

    # Performing the KNN model
    knn_model_2 <- knn(train = reduced.data[split != i, -ncol(reduced.data)],
                       test = reduced.data[split == i, -ncol(reduced.data)],
                       cl = reduced.data$Class[split != i],
                       k = j)  # Adjust k as needed

    # Predict the validation set
    predicted_knn_2 <- knn_model_2
    # Creating the confusion matrix
    conf_matrix_knn_2 <- confusionMatrix(as.factor(predicted_knn_2),
                                        as.factor(reduced.data$Class[split == i]))

    # Calculating the misclassification error
    knn_error2 <- 1 - conf_matrix_knn_2$overall["Accuracy"]
    error <- append(error, knn_error2)
  }

  error
  e <- unlist(error)
  mean_e<-mean(e)
  mean_e    # Storing the minimum error to a variable
  if (mean_e < min_error){
    min_error<-mean_e
    v <- j
  }
}
min_error
v
```