Predicting Invasiveness in Future Patients through Gene Expression Analysis

MA321-7 – Applied statistics

14th March 2024

Abstract

This report involves analyzing the gene expression data against the invasiveness of the gene combinations, using advanced statistical methods with machine learning techniques and the R language. It consists of crucial tasks such as data preprocessing and dimensionality reduction to make the process of analysis easy and efficient. Through techniques like PCA and LDA, the code effectively addresses challenges like high-dimensional datasets, revealing essential patterns and structures within the data. By reducing dimensionality while keeping the important information, these methods improve better understanding and interpretation of gene expression datasets. The model training phase displays flexible machine-learning algorithms. By evaluating these models using various metrics such as accuracy, the code ensures a complete evaluation of their performance using metricrelated plots, allowing for informed decision-making in model selection to predict the class of future

patients. Additionally, by using unsupervised learning methods like k-means clustering, the predictive models get better. This mix of methods combines the best of both supervised and unsupervised learning, making predictions more accurate and reliable. Overall, the report provides a valuable and efficient framework for analyzing gene expression data and developing predictive models for cancer invasiveness. Its systematic approach, from data exploration to model evaluation, empowers researchers to derive meaningful results and drive advancements in cancer diagnostics and treatment.

Table of Contents

Introduction

Gene expression analysis plays a crucial role in cancer diagnosis, prediction, and treatment, offering hope for better outcomes and quality of life for cancer patients. Through the analysis of gene expression using supervised learning and unsupervised learning, several important goals can be achieved, such as understanding disease mechanisms, predicting disease outcomes, drug invention and development, and clinical decision-making.

Performing several tasks in gene expression analysis enables the achievement of important objectives. Unsupervised dimension reduction techniques like Principal Component Analysis (PCA) and supervised

techniques like Linear Discriminant Analysis (LDA) help reduce the complexity of gene expression data while preserving essential information. This leads to improved data visualization, interpretation, and modeling, allowing for a better understanding of complex patterns and relationships among genes.

Unsupervised clustering methods allow the identification of clusters of genes and patients based on their similarities in gene expression profiles. This helps in uncovering hidden patterns and relationships within the data, providing deeper insights.

Supervised learning models predict whether patients have invasive or non-invasive cancer based on their gene expression profiles. Overall, these tasks form an extensive framework for analyzing gene expression data, from data exploration to predictive modeling, to advance the understanding of cancer biology and improve patient care.

Preliminary Analysis

Dimension of Dataset are 78 observations,2000 variables, and the outcome "Class."

- Observations represent patients.
- Variables represent the different gene expressions.
- Outcome "Class" indicates the invasiveness of cancer where Class 1 is invasive, and Class 2 is NonInvasive.

There are 34 patients with invasive cancer and 44 patients with Non-Invasiveness cancer.

Handling of missing data

There are overall **60** missing values in our dataset, which were handled by replacing the missing values with the meaning of the respective gene columns data.

Normalizing the data:

The range of Gene Expression Values has a minimum value of -2 and the Maximum value of 2.

While values appear to be centered around 0, normalization of data is considered essential due to the sensitivity of some techniques to scale.

Showing the first 10 rows of mean and SD

GeneID	Mean	SD
NM_007256	-0.0541	0.1904
NM_002048	-0.047	0.2376
NM_005746	-0.0451	0.1966
NM_021103	-0.0044	0.1446
NM_001197	-0.0359	0.259
NM_001504	-0.1319	0.3312
AB020636	-0.0554	0.1911
AB040937	-0.0429	0.1729
Contig40549_RC	-0.0357	0.2077
NM_001037	-0.0265	0.1919

Table 1: Mean and SD of first 10 genes.

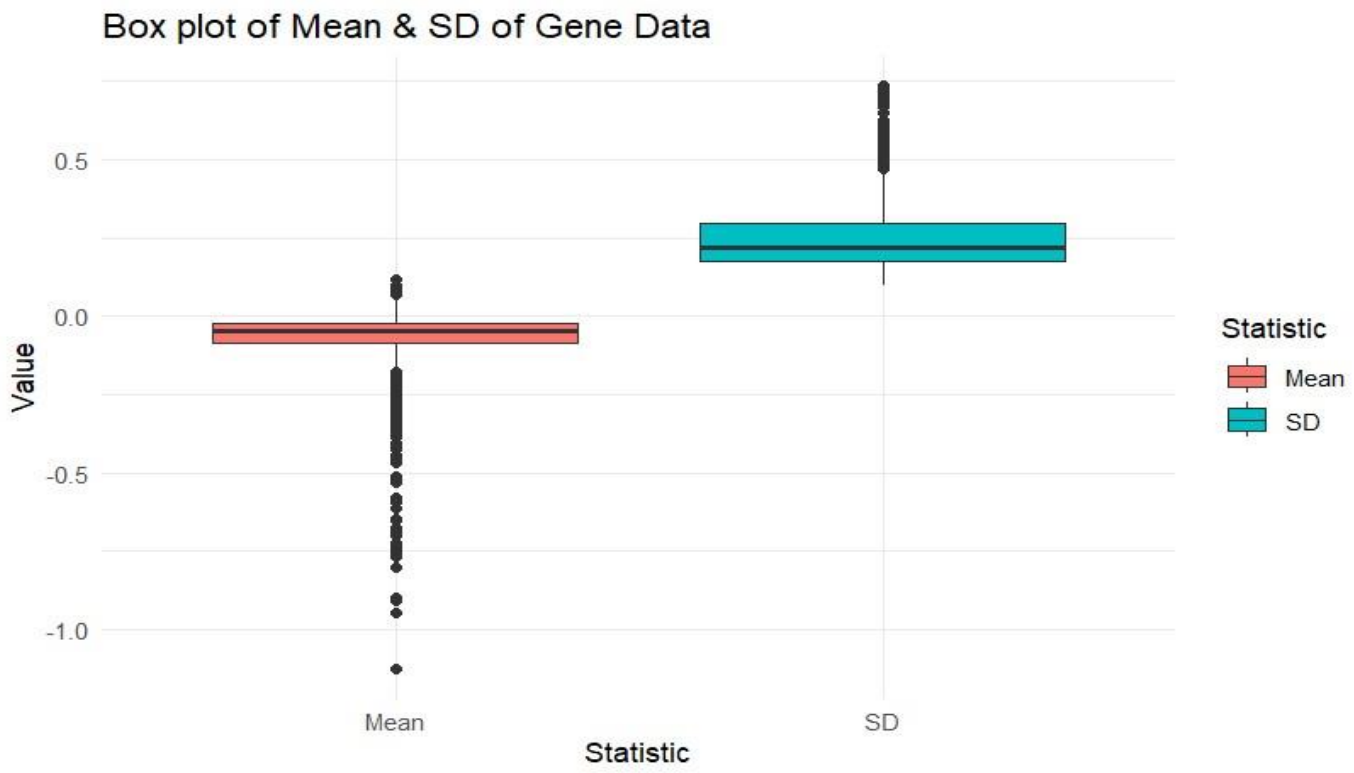

Figure 1: Box plot for Mean and SD of gene data.

From the plot of Mean and SD, the box for the mean is narrow and centered around zero, which suggests that the average expression levels across the genes are consistent and centered around a mean of zero. There are many outliers present below the mean box plot indicating that some genes have means that are lower than the average. The bold line inside the box indicates the median of the means, which is near zero.

The Box plot for the SD is wider than the mean, which implies that there is more variability in the standard deviations of gene expression levels than in the means. The SD box plot is also centered higher than zero, implying that there is a certain amount of variability in gene expression levels across patients. There are no outliers for the box for SD, so the variability across the genes doesn't have extreme deviations from the average.

Analysis

Dimensionality Reduction

A dataset of dimension (78,2000) is referred to as High Dimensional, and it poses various problems such as:

- Concerns of overfitting.
- High Collinearity between the genes
- Difficulty in Visualization
- Limitation of Analysis with High dimensional Data.

With so many predictors, it is important to identify the most relevant data and reduce the dimension to improve the performance of the model and its accuracy.

In this analysis, one Unsupervised and one supervised dimension were used to reduce the dimensionality of the large data.

Unsupervised Dimension Reduction Using PCA

PCA achieves Dimension Reduction by reducing the number of variables in a dataset while retaining most of the variability. It does this by transforming the original variables into a new set of uncorrelated variables called Principal Components. Observations of a single gene type tend to lie near each other in a lower dimensional space. Initially, it would have been difficult to visualize this in a dataset with 2000 variables and not know which one is more important. Knowledge of PCA components will enable us to visualize and understand the gene data.

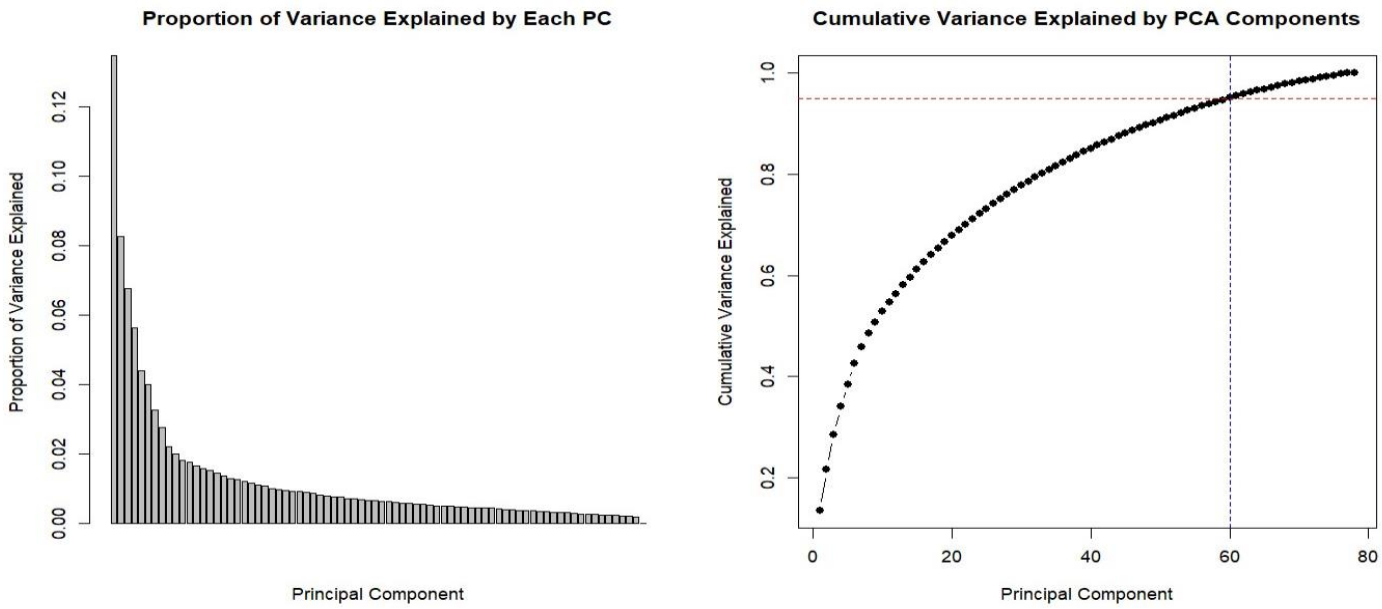

Figure 2: Proportion and Cumulative Variance Explained

The first few principal components from the Bar Plot can explain most of the data. PC1 represents 13.5 % of the variance and PC2 shows 8.3 % of the variance together it shows 21.8% of Variance.

These top principal components can be used for further analysis such as Clustering or training the models to predict future values. From the R code, it is derived that the first 60 Principal components represent 95% of the gene data is a huge improvement from 2000 variables.

Supervised Dimension Reduction Using LDA

LDA performs dimension reduction by projecting the data points onto a lower–dimensional space that maximizes the separation between the classes, effectively reducing the number of features used for classification.

The number of discriminant functions is equal to the number of classes minus one, and since there are only 2 classes, there will be only one discriminant function (LD1) in this case.

Figure 3: box plot showing the distribution of LD1 scores for each class.

The 2 classes can now clearly be separated using 1 Linear Discriminant Function instead of using 2000 variables.

Visualization is improved and prediction of new observations is easier.

Unsupervised learning models to investigate clusters

Principal Component Analysis (PCA)

By applying the cumulative variance function, 60 PCs were identified, which can collectively contain 95% of the total variance.

Component	Standard deviation	Proportion of Variance	Cumulative Proportion
PC1	16.4084	0.1346	0.1346
PC2	12.86128	0.08271	0.21732
PC3	11.62464	0.06757	0.28489
PC4	10.62124	0.05641	0.3413
PC5	9.38531	0.04404	0.38534

Table 2: Cumulative proportion and proportion of Variance of top 5 contributed PC

In the Scatter Plot of PC1 vs PC2, a cluster of points around the center of the plot suggests that many observations in the gene dataset have similar scores on both PC1 and PC2. These observations show that the patterns of gene expression variation captured by these two principal components.

The spread along the PC1 axis is wider compared to the PC2 axis, indicating that PC1 captures more variability in the data set. This implies that the genes contributing to PC1 are more influential expression changes compared to those contributing to PC2.

Feature Extraction:

Applied Feature Extraction technique using PCA to extract the most relevant genes associated with each principal component. Table 3 shows the Top 5 genes for each of the first 5 principal components. These genes have the most significant information among the datasets. Feature extraction helps in many ways by enhancing model performance and removing the less significant information.

PCS	Gene 1	Gene 2	Gene 3	Gene 4	Gene 5
PC 1	NM_016267	Contig50153_RC	Contig56678_RC	NM_001218	AB020689
PC 2	AF131817	Contig53881_RC	NM_004684	Contig54656_RC	NM_002742
PC 3	Contig12369_RC	NM_004297	Contig45347_RC	Contig5392_RC	Contig30061_RC
PC 4	Contig47796_RC	NM_003909	X60188	AL353957	NM_000788
PC 5	NM_003332	NM_001225	NM_003982	Contig372_RC	NM_007256

Table 3: Top 5 genes for each of the first 5 principal components

K-means Clustering

K-means clustering groups similar data points into k clusters. After performing the k-means function on the 60 PCs derived from PCA, with the optimal number of clusters, Figure 4 illustrates the visualization of k-means clustering for PC1 and PC2. Three clusters have formed. Red cluster classes may have a similar gene expression pattern that is distinct from those in clusters green and blue. Classes are spread along the negative side of PC1 and clustered around the origin of PC2. This might represent a specific state of gene expression or a particular group of conditions. The Green cluster classes are grouped, on the positive side of both PCs. The Blue Cluster

classes are spread along the negative side of PC2 and are distinct from the other two clusters, particularly along PC 2.

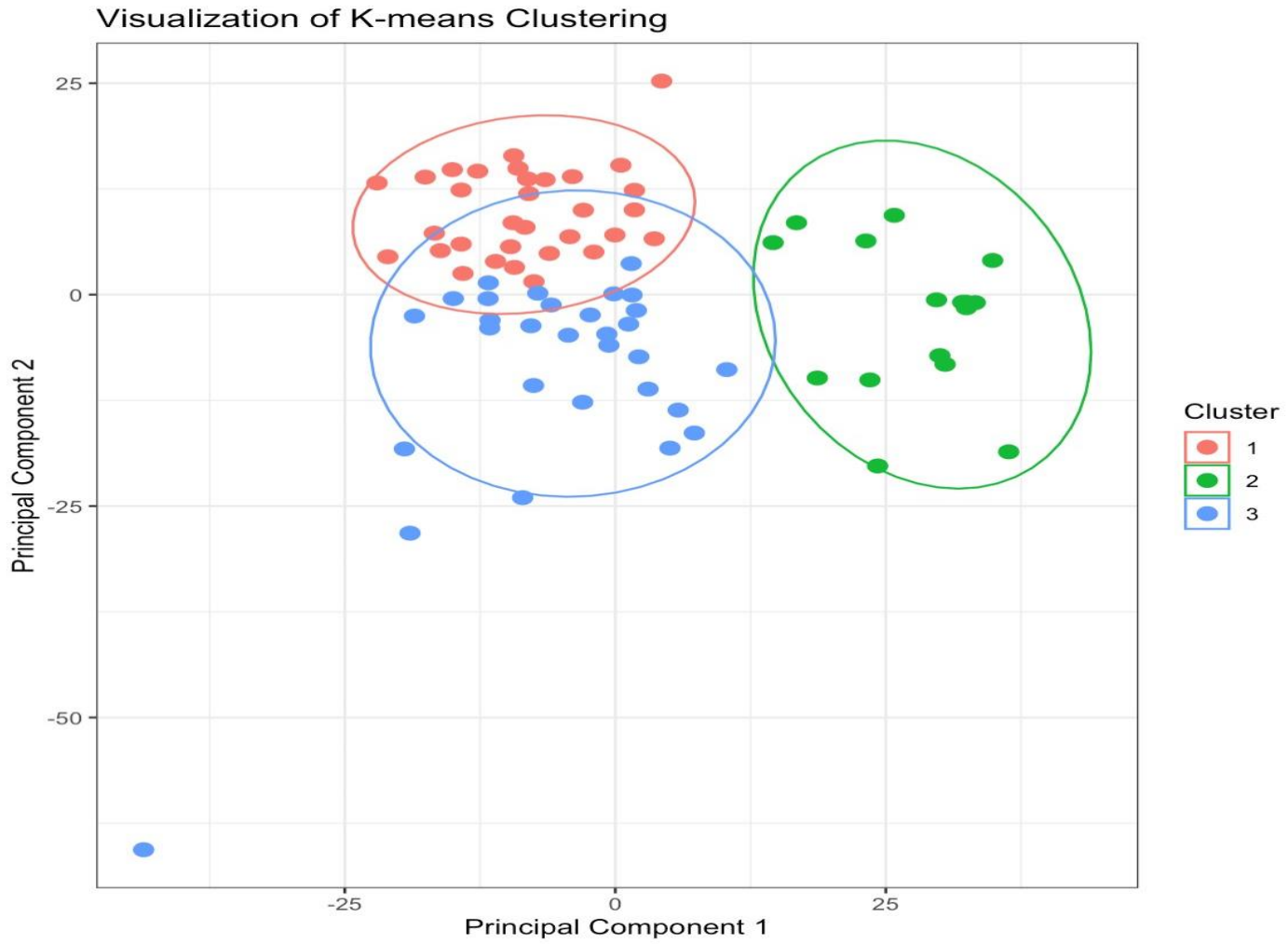

Figure 4: Ellipse plot of k-means Clustering

Hierarchical Clustering

The hierarchical clustering groups similar data points by constructing a hierarchy of clusters, which can be represented as a tree-like structure called a dendrogram.

For this analysis, Agglomerative Clustering is being considered where each data point is initially considered as a separate cluster. The algorithm then iteratively merges the closest pairs of clusters based on a distance or similarity measure until all data points are combined into a single cluster.

Considering the Average Linkage plot to visualize the clusters because the distance between two clusters is calculated by the average distance between all pairs of observation. The clustering process starts by considering each gene or class as a separate cluster and then progressively merges clusters until all genes or classes are in a single cluster. A lower height means the clusters are more similar and vice versa.

Figure 5 represents the balanced cluster and is less vulnerable to noise and outliers. 3 to 5 clusters have formed which can reveal sets of genes that are co-expressed under similar conditions. These clusters represent groups of genes that are likely to be involved in common pathways or regulatory networks.

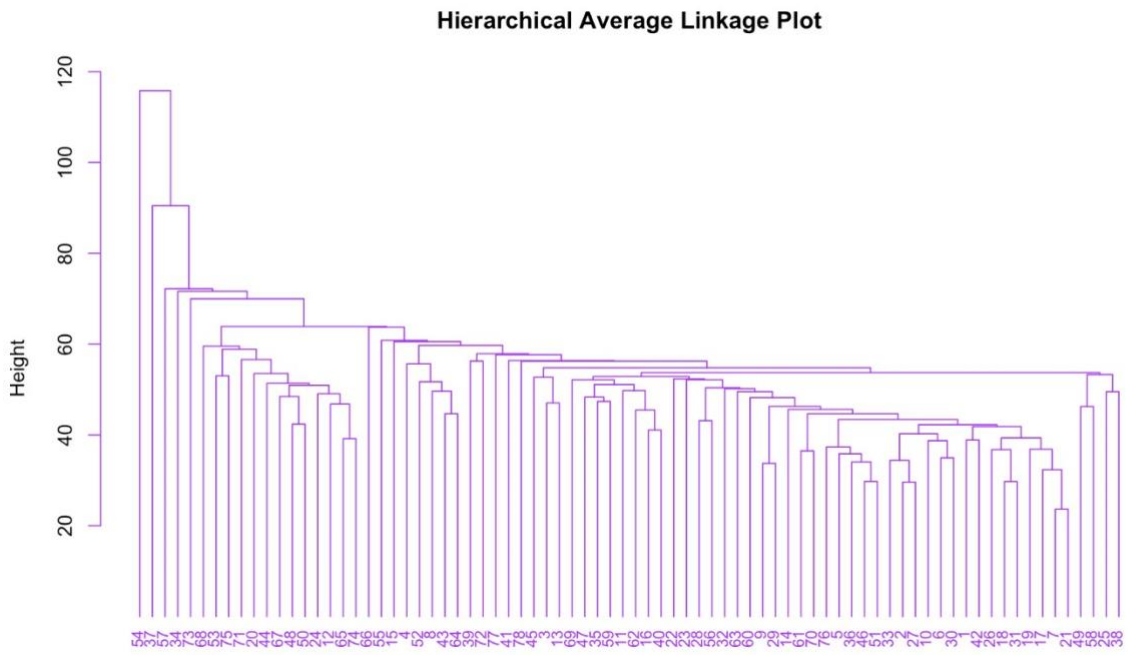

Figure 5: Average linkage plot for hierarchical clustering

Supervised learning models to predict the invasiveness

As the data preprocessing has already been done, the next step is to train the supervised learning models on the preprocessed data. The models under consideration are Logistic regression, LDA, QDA, k-NN, Random Forest, SVM, GBM, and Naïve Bayes. Along with training the models, using "train control", ensures that the model training and validation are done systematically which improves the reliability of the performance metrics and defines resampling methods and "tune grids" for hyperparameters with a set of values and it chooses the best values. This leads to a better-performing model when the new data is given to the model to predict the invasiveness.

Then, the data processed using PCA will be taken and divided into two parts. one for training the model and the other for testing the model. For this analysis, the data is divided into 80 percent training data and the remaining testing data.

The train control with 10-fold cross-validation is used for the whole analysis of the Supervised learning model to maintain consistency.

First, applying the **Logistic regression** model to the training data against the outcome class, with hyperparameters alpha and lambda in the tuning grid. Specific hypermeters are chosen to adjust the level of regularization applied to the model. The set of these two hyperparameters ensures that the model remains sensitive to detect the patterns in gene expression data and to avoid the main concern of overfitting issues.

The range has been 0 to 1 for alpha and 0.0001 to 1 for lambda are defined for these parameters so that the best pair of values can be chosen by the model while training what works best for the data and helps in deriving the well–tuned model that generalizes well to the testing and Future data.

For the **LDA** model, no hyperparameters were used because this model works on a fixed formula that doesn't change with any kind of parameter tuning.

For other models, considering 60 principal components data for the analysis, **QDA** expects a smaller number of columns to train, reduced the data to 26 PC columns, and applied a cross-validation strategy manually, which is good for assessing model performance. The code is already well-optimized for the capabilities of the QDA function without the usage of hyperparameters.

The **k-NN (k-Nearest Neighbors)** model uses k as a hyperparameter in tuning grid starting from 3 to 20 as a range of values by the step of 2, because if the k is starting from 1, then the prediction for the new data is based on the closest data point in the training set which can lead to overfitting and model can become too sensitive to

minor differences in the training data. Choosing an odd number for k, with step 2 ensures that there will always be a majority among the nearest neighbors and gives a clear win of one class for the classification of future data.

The **Random Forest** model's "mtry" is used for the tuning of this model because every gene doesn't contribute equally to the output. "mtry" increases the chances that each tree in the forest is involved in evaluating the relevant genes improving the model's ability to capture the patterns that are influencing the outcome.

The "mtry" is defined as a set of values that includes 2, the square root of the number of genes, half of the number of genes, and the total number of genes.

The set of values is arranged in a format where it ranges from limited to more explorative strategies for finding the best "mtry" value for the random forest model.

The **Support Vector Machine (SVM)** model is well known for its efficiency on high-dimensional data. The hyperparameters C and sigma were used in tuning this model with the set of values 0.1,1,10 and 0.01,0.1,1 respectively to decide on the pair of values which suits best.

The "C" value controls the balance between having a smooth decision boundary and accurately classifying training data points and influences how much you allow the SVM to tolerate misclassifying data. The values defined for the C help in identifying the sweet spot where the model is enough to capture the patterns without overfitting the training data.

The sigma value helps the model to decide how closely it pays attention to the training data. The smaller value of sigma makes the model sensitive to the data and the larger value makes the model seem smooth and doesn't react much to the minor details.

The **Gradient Boosting Machine (GBM)** builds the model in steps, adding a new model to correct the mistakes made by the previous models.

The hyperparameters n. trees, interaction. depth, shrinkage, and n. minobsinnode was used.

n. Trees with the set of values 50,100 and 150, define how many trees to build. A greater number of trees leads to better learning but with overfitting issues. The chosen set of values allows testing from a basic to a relatively high number of trees to see how model performance improves with more trees. interaction. Depth sets the maximum

depth of each tree, with values from 1,3,5, and 7 defined for our model. Shrinkage, which is also known as learning rate, controls how quickly the model learns about the training data.

n. minobsinnode specifies the minimum number of observations in the leaf nodes of the tree, which prevents the trees from growing too deep.

The combination of the values will be selected by the model which helps maintain the balance between the model complex enough to learn from the training data without fitting so closely that it performs poorly on testing or Future data.

The **Naive Bayes** model predicts the class of new data by looking at how often genes appear in each class if each gene doesn't influence the other.

3 hyperparameters are used for this model. Laplace, userkernel, and adjust.

Laplace with values 0,0.5 and 1, decides how much help is needed for the model to deal with new kinds of data. A value of 0 provides no help, 0.5 provides moderate help and 1 provides a lot of help making sure the model doesn't get confused by new information.

User Kernel with value True tells the model to use a flexible approach that can adapt better to the data's shape while false sticks with a simple and more straightforward method. adjust with value 1 sees the data as it is and 2 is making the data's ups and downs smoother and less rough and making the model less sensitive to small details.

Once all the models are done with training on the train data with their respective hyperparameters and 10-fold cross-validation, resampling needs to be done to compare the models and to find the best model.

Below are the results for all the models for the metric accuracy:

Model Name	**Median Accuracy**	**Mean Accuracy**	**Median Kappa**	**Mean Kappa**
Logistic Regression	0.5714	0.6095	0.1546	0.1998
Linear Discriminant Analysis[LDA]	0.5357	0.5595	0.1235	0.1291
Quadratic Discriminant Analysis[QDA]	0.5982	0.5894	0.1592	0.1643
k-Nearest Neighbours [k-NN]	0.6667	0.6524	0.3333	0.2302
Random Forest	0.5714	0.519	0.0435	-0.0208
Support Vector Machine [SVM]	0.5714	0.631	0.1546	0.2468
Gradient Boosting Machine [GBM]	0.619	0.6238	0.2467	0.2507
Naive Bayes	0.6667	0.5524	0.2917	0.1014

Table 4: Comparison of Accuracy and Kappa of Supervised Learning Models

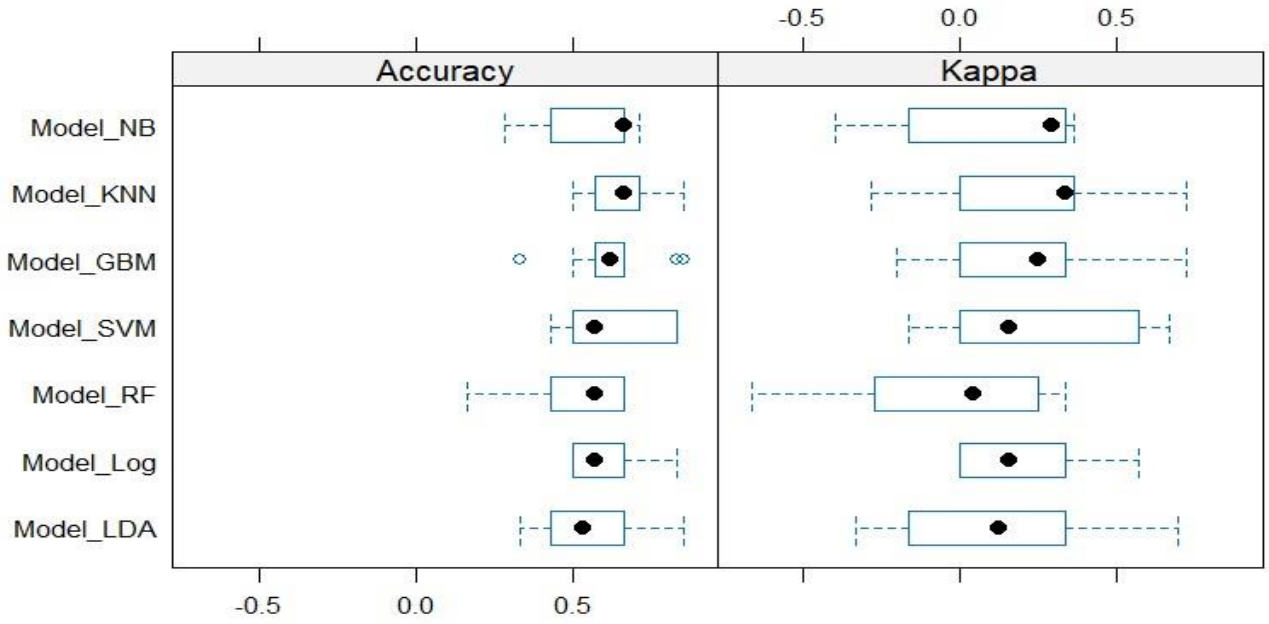

Figure 6: Box-whisker plot for Accuracy and Kappa of Supervised learning models

Based on the table above and the plot, the k-NN and the Naïve Bayes model have a better accuracy of value **0.66** when compared to all the other models. In this scenario, the values for the Kappa metrics need to be considered to derive the best model because Kappa compares the observed accuracy of a model against the accuracy that could be expected by chance. From the table, the k-NN model has a better Kappa value of **0.33** which is greater than the Naïve Bayes's Kappa value of 0.29.

Thus, we are concluding the **k-NN is the best model** derived based on accuracy and kappa metric from resampling results.

Now, the best models need to be improved by including the clusters formed from the k-means clustering labels as a new predictor. Split the data into 80% train and 20% test again and evaluate the impact of this change using the k-NN model with an optimal k value from the model that is previously trained. Training with these labels aims to improve the model's predictive accuracy by taking advantage of additional information provided by the clustering.

The accuracy of the model is evaluated on the testing set by creating a confusion matrix to compare the model's predictions against the actual classes. The improved model's accuracy is **0.79**. The results indicate that

incorporating clustering information into the k-NN model resulted in an improved accuracy rate compared to the original model. This suggests that the clusters identified through unsupervised learning contribute valuable information for enhancing the performance of our machine-learning model.

Discussion and Conclusion

The analysis was conducted utilizing a mix of statistical methods and machine-learning tools using the existing data of genes for the patients whose classes are already defined. The k-NN model stands out as the best model to predict the invasiveness of cancer with an accuracy of 79%. This means that the k-NN model can identify the invasiveness correctly in nearly 8 out of 10 patients based on the information it learned from the previous cases which is less guessing and more knowing.

This approach is like finding a shortcut that still gets us to the right place most of the time, making it a valuable tool for doctors trying to understand and treat cancer more effectively. This is a big step forward because it helps the health sector to pinpoint who needs more urgent and special care, making sure that the right treatment can be given to those who need it most, faster, and more accurately than before.

References

[1] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2023). *Introduction to Statistical Learning, Second Edition* (Corrected June 2023 ed.)

[2] Jolliffe, I. T. (2002*). Principal component analysis.* Wiley Online Library.

[3] Hastie, T., Tibshirani*,* R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction.* Springer Science & Business Media.

[4] Bishop, C. M. (2006*). Pattern recognition and machine learning.* springer.

[6] Raschka, S., & Mirjalili, V. (2019). *Python Machine Learning.* Packt Publishing Ltd.

Appendix

```
## This R code is for Team E from Group 5##

# Clear previous history of images and variables

if(!is.null(dev.list())) dev.off() rm(list = ls())

# Libraries needed for our analysis library(MASS)

library(stats)

library(ggplot2)

library(caret)

library(glmnet)

library(Metrics)

library(randomForest)

library(kernlab)

library(gbm) library(rpart)

library(cluster)

library(e1071)

library(dplyr)

library(knitr)

library(ggfortify)

library(factoextra)
```

```
library(ggdist)

library(class)

library(kableExtra)


# Setting up the working directory

#setwd("C:/Users/cc/OneDrive - University of Essex/MA321/Group")#Kago setwd("C:/Users/Avinash

Reddy/Videos/Applied - GRP-Assgn")#prasanna

# Read data from the CSV file

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

#setting Random seed set.seed(2314558)

G5TE_Data_subset <- rank(runif(1:4948))[1:2000]

# Fetching data of the 2000 columns

G5TE_Data_Analyze <- Given_Data[, G5TE_Data_subset]

#### Preliminary Analysis ####

# analyze the structure and dimensions of the given data

str(G5TE_Data_Analyze) dim(G5TE_Data_Analyze) #

checking for missing values

sum(is.na(G5TE_Data_Analyze))

# handling missing values by replacing them with mean of the respective column

# Looping through each column in the Given data

for(i in 1:ncol(G5TE_Data_Analyze)) {   #
```

```
Checking if the column is numeric

if(is.numeric(G5TE_Data_Analyze[[i]])) {

   # Calculating mean of the column without NA values

column_mean <- mean(G5TE_Data_Analyze[[i]], na.rm = TRUE)

# Replacing NA values with the column mean


   G5TE_Data_Analyze[[i]][is.na(G5TE_Data_Analyze[[i]])] <- column_mean

  }

}

# check the range of gene expression values

# Get the range for all gene expreesion values gene_expression_range

<- range(unlist(G5TE_Data_Analyze)) print(gene_expression_range)

# Data Normalization

G5TE_Data_Normalized <- scale(G5TE_Data_Analyze)

#Plotting Mean and SD

# Initialize vectors to store the mean and SD for each gene

Mean_G5TE = numeric(ncol(G5TE_Data_Analyze))

SD_G5TE = numeric(ncol(G5TE_Data_Analyze))

# Loop through each gene to calculate mean and SD for

(i in 1:ncol(G5TE_Data_Analyze)) {

  Mean_G5TE[i] = mean(G5TE_Data_Analyze[,i ])

  SD_G5TE[i] = sd(G5TE_Data_Analyze[,i])
```

```
}
# Combine into a data frame
DF_Mean_SD <- data.frame(GeneID = colnames(G5TE_Data_Analyze),
                         Mean = Mean_G5TE, SD = SD_G5TE)
# View the first 10 rows head(DF_Mean_SD,
10) # Boxplot for Mean and SD of the gene
expression data

DF_stats = data.frame(Value = c(Mean_G5TE, SD_G5TE),
                      Statistic = factor(rep(c("Mean", "SD"),
each = ncol(G5TE_Data_Analyze)))) ggplot(DF_stats, aes(x =
Statistic, y = Value, fill = Statistic)) +   geom_boxplot() +
labs(title = "Box plot of Mean & SD of Gene Data",
      x = "Statistic",
y = "Value") +
theme_minimal()
#Maintaining separate data frame for data with column Class G5TE_DataWithClass
<- G5TE_Data_Analyze
G5TE_DataWithClass$Class <- as.factor(Given_Data$Class)
# Investigate the class column table(G5TE_DataWithClass$Class)
################ Code for Task 1 ###################
## unsupervised Dimension reduction using PCA ##
```

```
# using prcomp() to get principal component
G5TE_PCA_Result <- prcomp(G5TE_Data_Normalized ,scale. = TRUE,center = TRUE)
summary(G5TE_PCA_Result)
# Scatter plot of first 5 PC components cols <-
ifelse(G5TE_DataWithClass[,2001]=="1", 'blue', 'red')
# Save current par settings to reset later old_Par
<- par(no.readonly = TRUE)

# Increase the upper margin to make space for the title par(mar=c(5.1,
4.1, 4.1, 2.1))
# Create pairs plot pairs(G5TE_PCA_Result$x[,1:5],
pch=19, cex=.5, col=cols)
# Add main title using mtext mtext("Scatterplot Matrix of the First Five Principal
Components", side=3, line=3, cex=1)
# Reset to old par settings par(old_Par)
# Calculating proportion of variance
G5TE_PropVar <- G5TE_PCA_Result$sdev^2 / sum(G5TE_PCA_Result$sdev^2)
# Calculating cumulative variance explained G5TE_CumVar
<- cumsum(G5TE_PropVar)
# Creating a summary data frame for PC data G5TE_PCA_Summary
<- data.frame(
  G5TE_PCASum_SD = G5TE_PCA_Result$sdev,
```

```
  G5TE_PCASum_PV = G5TE_PropVar,
  G5TE_PCASum_CV = G5TE_CumVar)
# Bar plot for the proportion of variance explained by each PC par(mfrow
= c(1, 2))
barplot(G5TE_PCA_Summary$G5TE_PCASum_PV, names.arg = G5TE_PCA_Summary$PC,
        xlab = "Principal Component", ylab = "Proportion of Variance Explained",
main = "Proportion of Variance Explained by Each PC")
# Selecting only the first 10 components for display as table G5TE_PCA_top10
<- head(G5TE_PCA_Summary, 10) kable(G5TE_PCA_top10, format =
"html", digits = 3) %>%   kable_styling()
# Finding the number of components needed to explain 95%
Ncomp <- which(G5TE_CumVar >= 0.95)[1]
# Plotting the cumulative variance explained to visualize
plot(G5TE_CumVar, xlab = "Principal Component",       ylab =
"Cumulative Variance Explained",       type = "b", pch = 19,
main = "Cumulative Variance Explained by PCA Components")
abline(h = 0.95, col = "red", lty = 2) abline(v = Ncomp, col =
"blue", lty = 2) cat("Number of principal components to keep:",
Ncomp, "\n") # Fetching dimension reduced data of PCA
G5TE_PCA_Reduced <- G5TE_PCA_Result$x[, 1:Ncomp]
## supervised Dimension reduction using LDA ##
```

```
LDA_Predictors <- G5TE_Data_Analyze

LDA_Outcome <- Given_Data$Class

# Fit the LDA model lda_result <- lda(LDA_Outcome ~ .,

data = LDA_Predictors)

#  Fetch the transformed data from LDA lda_reduced_data

<- predict(lda_result, LDA_Predictors)$x

# Convert the data into data frame lda_reduced_data <-

as.data.frame(lda_reduced_data)

lda_reduced_data$Class <- as.factor(Given_Data$Class)

# Box Plot for LDA reduced data lda_boxplot <-

ggplot(lda_reduced_data, aes(x = Class, y = LD1, fill =

Class)) +   geom_boxplot() +   labs(title = "LDA:

Boxplot of LD1 by Class", x = "Class", y = "LD1") +

theme_minimal() print(lda_boxplot)

################ Code for Task 2 ################

## unsupervised learning models/clustering ##

## Principal Component Analysis ##

#extract the scores of the first 60 principal components  G5TE_PCA_Result$x

<- G5TE_PCA_Result$x[, 1:60]

#performing standard deviation for 60 pcs

G5TE_PCA_Result$sdev <- G5TE_PCA_Result$sdev[1:60]
```

```
#loading the information of 60 components, it will have only information of 60 pcs

G5TE_PCA_Result$rotation <- G5TE_PCA_Result$rotation[, 1:60]

#create dataframe for 60 pcs

G5TE_Data_Reduced <- data.frame(G5TE_PCA_Reduced)

#add Class names to dataframe

G5TE_Data_Reduced$Class <- rownames(G5TE_Data_Analyze)

# plot PC1 Vs. PC2 ggplot(G5TE_Data_Reduced, aes(x

= PC1, y = PC2)) +   geom_point(aes(color = Class),

size = 3) +     xlab("PC1") + ylab("PC2") +

ggtitle("PCA Plot (PC1 vs PC2)")


# Get the data loadings for the first 60 principal components load_components<-

G5TE_PCA_Result$rotation[, 1:60]

# Initialize a list to store the top genes/values for each component top_feature_value

<- vector("list", length = ncol(load_components))

# Loop through each principal component for (i

in seq_along(top_feature_value)) {   # Get the

gene scores for the current component

feature_score <- abs(load_components[, i])

 # Get the top 10 genes/values for the current component

feature_rank <- sort(feature_score, decreasing = TRUE)

high_feature <- names(feature_rank[1:10])  # Store the
```

```
top genes/values in the list   top_feature_value[[i]] <-
high_feature
}
# Print the top genes/values for each component for (i in seq_along(top_feature_value)) {   cat("Top 10
genes for Principal Component", i, ":", paste(top_feature_value[[i]], collapse = ", "), "\n") }
# Extract the top 5 genes/values for the first 5 principal components high_feature_matrix
<- sapply(top_feature_value[1:5], function(x) x[1:5])
# perform table creation high_feature_df <-
data.frame(t(high_feature_matrix))
# Add row and column names row.names(high_feature_df)
<- paste("PC", 1:5) colnames(high_feature_df) <- 1:5

# Display the table kable(high_feature_df, caption = "Top 5 genes/values for each of the first 5
principal components")
## K-means Clustering ##
#Plotting wss method to find the desired number of clusters fviz_nbclust(G5TE_PCA_Result$x
, kmeans, method = "wss")
#choosing desired clusters desired_clusters <- 3  #perform K-means for 60 pcs
kmeans_perform <- kmeans(G5TE_PCA_Result$x , centers = desired_clusters)
# Get cluster labels cluster_title <-
kmeans_perform$cluster
```

```
# Create a data frame with PC1, PC2, and cluster title kcluster_df

<- data.frame(

  PC1 = G5TE_PCA_Result$x[, 1],

  PC2 = G5TE_PCA_Result$x[, 2],

  Cluster = as.factor(cluster_title)

)

# Ellipse plot for Visualization of K-means Clustering kmeans_plot <-

ggplot(kcluster_df, aes(x = PC1, y = PC2, color = Cluster)) +

geom_point(size = 3) +   stat_ellipse(aes(group = Cluster), geom = "polygon",

fill = NA, alpha = 0.5) +   xlab("Principal Component 1") + ylab("Principal

Component 2") +   ggtitle("Visualization of K-means Clustering") +


  theme_bw() #

Display the plot

print(kmeans_plot)

## Hierachical ##

# Hierarchical clustering with different linkage methods

Hierarchical_average <- hclust(dist(G5TE_PCA_Result$x), method = "average")

#create different colours for each linkage methods average_color <-

"purple" # plot dendrogram plot plot_dendrogram <-

function(dendrogram, color, main_title) {   plot(dendrogram, hang = -

1, cex = 0.8, main = main_title, col = color) }
```

```
# Average Linkage Dendrogram plot par(mfrow = c(1, 1))

plot_dendrogram(Hierarchical_average, average_color, "Hierarchical Average Linkage Plot")

##############Code for task 3 ################ ##

supervised learning models/classification ##

G5TE_Data_Reduced$Class <- Given_Data$Class

G5TE_Data_Reduced$Class <- as.factor(G5TE_Data_Reduced$Class)

#seeding again to maintain consistency set.seed(2314558)

#Splitting reduced Data into 80 percent training data and 20 percent test data

Reduced_Split_Index <- createDataPartition(G5TE_Data_Reduced$Class, p = 0.8,

                          list = FALSE,  times = 1)

G5TE_Training <- G5TE_Data_Reduced[ Reduced_Split_Index,]

G5TE_Testing  <- G5TE_Data_Reduced[-Reduced_Split_Index,]

G5TE_Training$Class <- as.factor(G5TE_Training$Class)

# cross validation with 10 folds

Tr_ctrl_Com <- trainControl(method="cv", number=10)

# tune grid with hyper parameters for log model log_TG

<- expand.grid(   alpha = seq(0, 1, by = 0.1),   lambda =

seq(0.0001, 1, by = 0.01)

)

# training Logistic regression model  set.seed(2314558)

G5TE_Train_Log <- train(Class ~ ., data = G5TE_Training,
```

```
                method = "glmnet",
family = "binomial",
trControl = Tr_ctrl_Com,
tuneGrid = log_TG)
# Summary of the trained log model
summary(G5TE_Train_Log) #
training LDA model
set.seed(2314558)
G5TE_Train_LDA <- train(Class ~ ., data = G5TE_Training,
                method = "lda",
preProcess = c("center", "scale"),
trControl = Tr_ctrl_Com

)


# Summary of the trained log model summary(G5TE_Train_LDA)
# Reducing the data furthermore to train QDA model G5TE_PC_QDA
<- G5TE_PCA_Result$x[, 1:26]
G5TE_PC_QDA_DF <- data.frame(G5TE_PC_QDA)
G5TE_PC_QDA_DF$Class <- Given_Data$Class
G5TE_PC_QDA_DF$Class <- as.factor(G5TE_PC_QDA_DF$Class)
#seeding again to maintain consistency set.seed(2314558)
```

```
# Create 10 equally sized folds
G5TE_QDA_Fold <- createFolds(G5TE_PC_QDA_DF$Class, k = 10, list = TRUE)
# Initialize a vector to store accuracy for each fold accuracy_QDA
<- numeric(length(G5TE_QDA_Fold))
# Loop through the folds for(i in
seq_along(G5TE_QDA_Fold)) {
  # Split the data into training and test sets
  G5TE_Training_QDA <- G5TE_PC_QDA_DF[-G5TE_QDA_Fold[[i]], ]
  G5TE_Testing_QDA <- G5TE_PC_QDA_DF[G5TE_QDA_Fold[[i]], ]
  G5TE_Train_QDA <- qda(Class ~ ., data = G5TE_Training_QDA)
  G5TE_Pred_QDA <- predict(G5TE_Train_QDA, G5TE_Testing_QDA)   set.seed(2314558)
  # Fit the QDA model to the training data
  G5TE_Train_QDA <- qda(Class ~ ., data = G5TE_Training_QDA)
  # Make predictions on the test set
  G5TE_Pred_QDA <- predict(G5TE_Train_QDA, G5TE_Testing_QDA)
  G5TE_ConfMat_QDA <- table(Predicted = G5TE_Pred_QDA$class, Actual = G5TE_Testing_QDA$Class)
accuracy_QDA[i] <- sum(diag(G5TE_ConfMat_QDA)) / sum(G5TE_ConfMat_QDA)
}
# Calculate mean and median accuracy mean_acc_QDA
<- mean(accuracy_QDA) median_acc_QDA <-
median(accuracy_QDA)
```

```
# Summarize the QDA model

summary(G5TE_Train_QDA) # Train k-NN

Model knn_TG <- expand.grid(k = seq(3, 20,

by = 2)) set.seed(2314558)

G5TE_Train_KNN <- train(Class ~ ., data = G5TE_Training,

                method = "knn",

trControl = Tr_ctrl_Com,

tuneGrid = knn_TG

)

# Summarize the K-NN model summary(G5TE_Train_KNN)

# Train random forest Model

Mtry_Ranfor <- c(2, sqrt(ncol(G5TE_Training)-1), (ncol(G5TE_Training)-1)/2, ncol(G5TE_Training)-1)

Ranfor_TG <- custom_tune_grid <- expand.grid(mtry = Mtry_Ranfor) set.seed(2314558)

G5TE_Train_RanFor <- train(Class ~ ., data = G5TE_Training,

                method = "rf",

trControl = Tr_ctrl_Com,

tuneGrid = Ranfor_TG) #

Summarize Random Forest  model

summary(G5TE_Train_RanFor)

# Train SVM Model

TG_SVM <- expand.grid(
```

```
  C = c(0.1, 1, 10),
sigma = c(0.01, 0.1, 1)
)
set.seed(2314558)
G5TE_Train_SVM <- train(Class ~ ., data = G5TE_Training,
                  method = "svmRadial",
trControl = Tr_ctrl_Com,
tuneGrid = TG_SVM)
# Summarize the SVM Model
summary(G5TE_Train_SVM) #
Train GBM Model
TG_GBM <- expand.grid(
  n.trees = c(50, 100, 150),
  interaction.depth = c(1, 3, 5, 7),
shrinkage = c(0.01, 0.1, 0.2),
  n.minobsinnode = c(5, 10, 20)
)
set.seed(2314558)
G5TE_Train_GBM <- train(
  Class ~ .,    data =
G5TE_Training,    method
```

```
= "gbm",   trControl =
Tr_ctrl_Com,   tuneGrid =
TG_GBM
)
# Summarize the GBM Model summary(G5TE_Train_GBM)
# Train Naive Bayes Model TG_NB
<- expand.grid(
  laplace = c(0, 0.5, 1),
usekernel = c(TRUE, FALSE),
  adjust = c(1, 2)
)
set.seed(2314558)
G5TE_Train_Nai_Ba <- train(Class ~ ., data = G5TE_Training,
                 method = "naive_bayes",
trControl = Tr_ctrl_Com,                  tuneGrid
= TG_NB)

# Summarize the NB Model summary(G5TE_Train_Nai_Ba)
#Combine models for comparison G5TE_Comp_Models
<- list(
  Model_Log = G5TE_Train_Log,
  Model_LDA = G5TE_Train_LDA,
```

```
  Model_KNN = G5TE_Train_KNN,
  Model_RF = G5TE_Train_RanFor,
  Model_SVM = G5TE_Train_SVM,
  Model_GBM = G5TE_Train_GBM,
  Model_NB = G5TE_Train_Nai_Ba
)
# Comparing Using resampling
G5TE_resam_res <- resamples(G5TE_Comp_Models)
# Summarize the results on accuracy Metric
G5TE_Acc_res <- summary(G5TE_resam_res, metric = "Accuracy") print(G5TE_Acc_res)
# Summarize the results on accuracy Metric
G5TE_kap_res <- summary(G5TE_resam_res, metric = "Kappa") print(G5TE_kap_res)
#Plot the accuracy of trained models from re-sampling results bwplot(G5TE_resam_res)

########## Code For Task 4 #################
# Add the cluster labels to your original data set as a new feature
G5TE_Data_Reduced$Cluster <- kmeans_perform$cluster
G5TE_Data_Reduced$Cluster <- as.factor(G5TE_Data_Reduced$Cluster)
# setting Training and Testing Data for the Best Model set.seed(2314558)
trainIndex_best <- createDataPartition(G5TE_Data_Reduced$Class, p = 0.8,
                                       list = FALSE,
times = 1)
```

```
G5TE_Training_best <- G5TE_Data_Reduced[ trainIndex_best,]

G5TE_Testing_best  <- G5TE_Data_Reduced[-trainIndex_best,]

#Fetching the Best value for K from the KNN Model

KNN_K <- G5TE_Train_KNN$bestTune$k KNN_K

set.seed(2314558)

# Train the k-NN model using the best k

G5TE_KNN_Best <- knn(train = G5TE_Training_best[, -
which(names(G5TE_Training_best)  %in% c("Class"))], test = G5TE_Testing_best[, -
which(names(G5TE_Testing_best) %in% c("Class"))], cl = G5TE_Training_best$Class, k =
KNN_K)

# Compute the accuracy of the improved model

G5TE_Best_ConfMat <- confusionMatrix(G5TE_KNN_Best, G5TE_Testing_best$Class)

G5TE_Best_Acc <- G5TE_Best_ConfMat$overall['Accuracy'] print(paste("The
Improved best Model's accuracy with clusters data:", G5TE_Best_Acc))
```

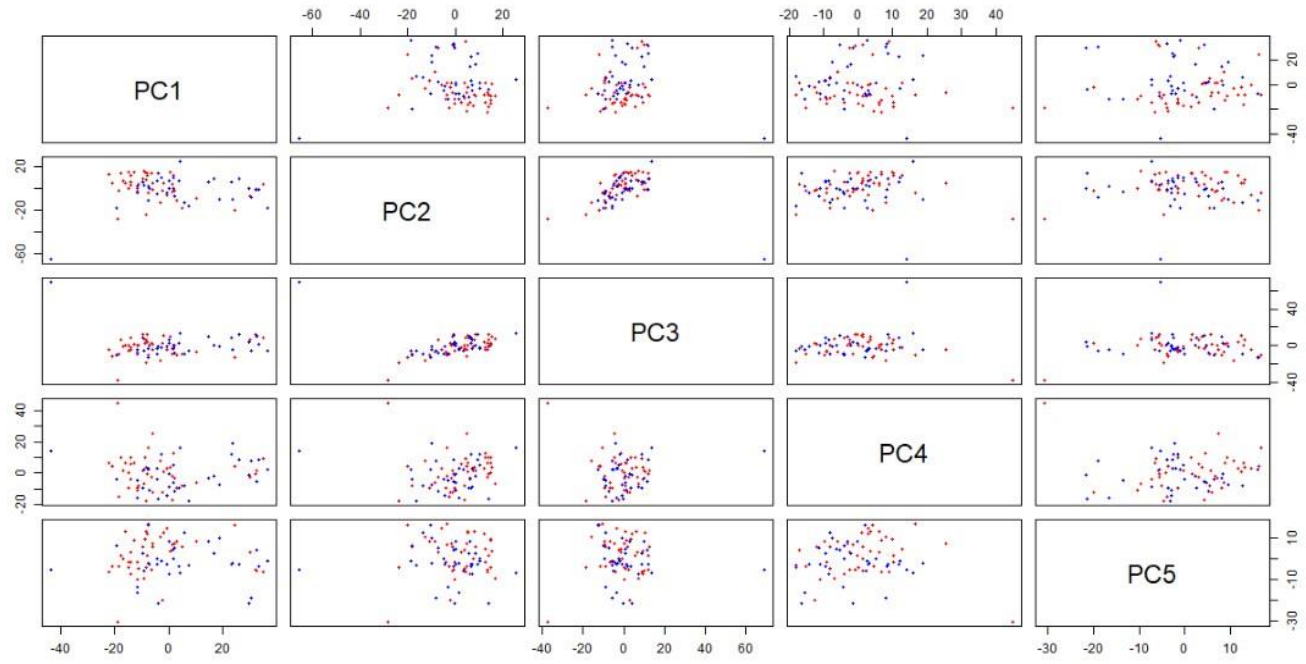

Figure 7: Scatter plot for first 5 principal components

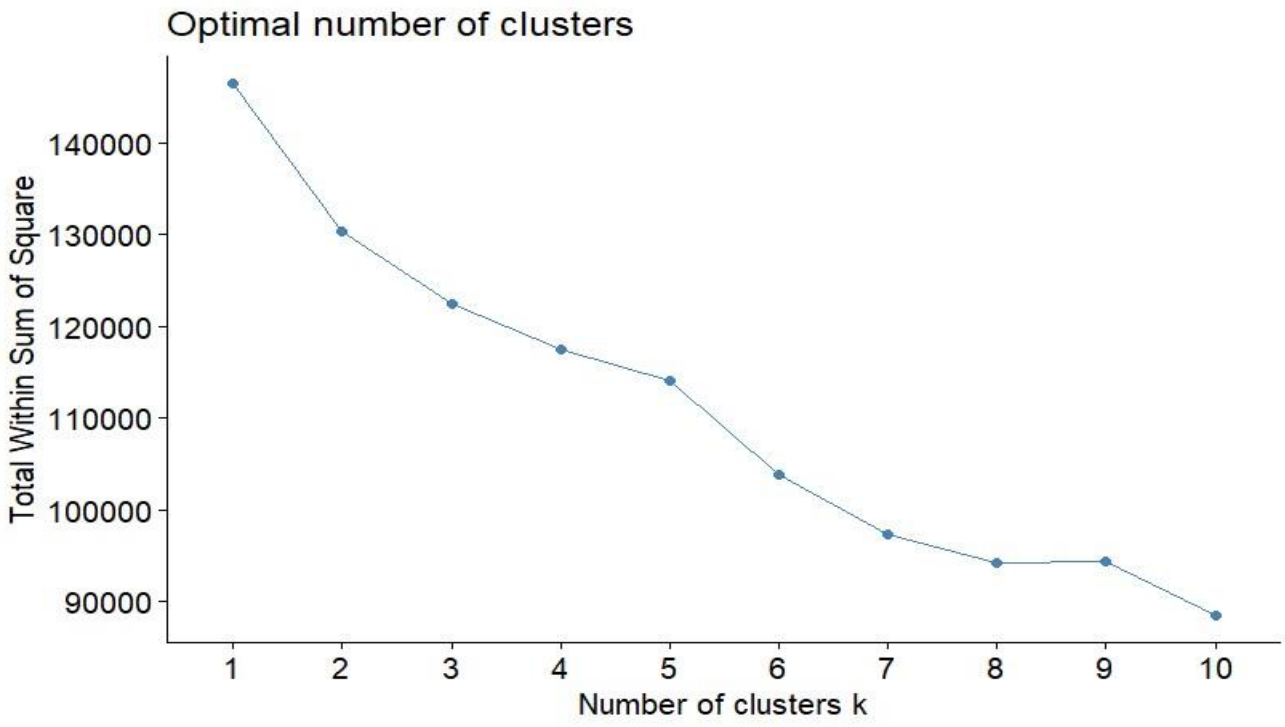

Figure 8: Elbow method plot for the number of clusters

Name	ID	Contribution to the Group Assignment
Kago Ronald Thipe	2310567	Worked on R code for Dimensionality reduction and prepared content for Task 1 analysis and preliminary analysis and helped with report completion
Nithyasree Velayutham	2310618	Worked on unsupervised machine learning models in the form of both R code and Analysis which was included in the report. Worked on Report's Abstract and Introduction and helped with report completion
Lakshmi Prasanna Reddy Tiyyagura	2314558	Worked on Supervised machine learning models for both R code and analysis that was put in the report. R code for preliminary analysis. did the analysis part for task 4 and wrote the conclusion. Finalized the report by merging all the team mate's analysis and merged the code into one.
Anand Kumar Srinivas	2309755	Worked on Task - 4 R code.In Report , worked on Task 3 analysis,Abstract and Introduction and helped with report completion
Shahbaz Sharif	2310201	Worked on Task - 4 R code and helped with report completion