Gene Expression Analysis for Cancer Classification

Author: Virendrasinh Narendrasinh Chavda
School of Mathematics, Statistics and Actuarial Sciences,
University of Essex, UK

Abstract

Early detection of invasive cancer is crucial for determining effective treatment plans. This project investigates the use of statistical and machine learning methods to classify cancer types based on gene expression data. Given the high dimensionality of gene expression datasets, this study explores various dimensionality reduction techniques including two-sample t-tests, LASSO regression, and variance-based feature selection. Missing data imputation was performed using k-Nearest Neighbors (kNN) and median imputation methods. The reduced datasets were then used to train several supervised machine learning models such as K-Nearest Neighbors (KNN), Support Vector Machines (SVM), and Logistic Regression. Resampling techniques like K-fold cross-validation and bootstrapping were employed to validate model performance and prevent overfitting.

The results demonstrated that combining two-sample t-tests with LASSO regression yielded significant improvements in classification accuracy. Specifically, the KNN and SVM models achieved near-perfect classification performance on the reduced feature set. Unsupervised learning methods, including clustering and Principal Component Analysis (PCA), were also applied to explore the inherent structure of the data. This analysis provides valuable insights into effective dimensionality reduction strategies for high-dimensional biological data and their impact on predictive modeling.

Contents

1. Introduction

The analysis of cancer diagnosis is crucial for understanding the recommended course of treatment for patients. Accurate analysis is key in this task, and innovative approaches to clinical data analysis, such as machine learning and big data tools, have become indispensable. By leveraging supervised and unsupervised learning models, researchers can uncover patterns in data that may aid in determining statistical relevance between genes, thus facilitating cancer type prediction.

In this study, gene expression data from cancer patients serves as predictors, with the invasive and non-invasive cancer as dependent variable. After sampling the data with the largest registration number of the team, 2000 random columns were chosen for further analysis. This high dimensionality in data makes it difficult to identify the genes which are decisive in classifying cancer types. To solve this, dimensionality reduction were employed. Specifically, three dimensionality reduction techniques—Two sampled t-test, LASSO regression, and feature reduction using the genes with more variance—were used to test the models. The aim was to compare the performance of supervised models combined with different reduction techniques.

Section 2 of this report provides a detailed overview of the gene expression data used in the study. Section 3 considers dimensionality reduction strategies, implementation of supervised learning models for classification, resampling methods, and the performance metrics used to evaluate the models. In Section 4, the findings of the study are discussed.

This section highlights cross validated and bootstrap validated performance comparison of different supervised models using data reduced after different reduction methods. Finally, Section 5 summarizes the key findings and implications of these findings in classifying types of cancer using gene expression data.

2. Preliminary Analysis

The dataset includes 4949 columns, 4948 columns represent genes and 1 column represent label "Class" which is divided into classes 1 (invasive) and 2 (non-invasive). The rows are patients and each entry in a row is the expression of a particular gene in each column [3]. First, 2000 columns were selected randomly after setting largest registration number from the group (2315880) as seed value, and a sample is created using these 2000 columns, and column "Class" is added in this sampled data.

*Dimensions of original dataset = 78 rows * 4949 columns*

*Dimensions of the sampled data = 78 rows * 2001 columns*

2.1. Dealing with missing values

It was found that there were 74 missing values in the sampled dataset. Out of these 74 missing values, row 54 had 73 missing values and row 23 had one missing value. An experiment was conducted to determine whether row 54 should be dropped as it is very highly likely that these missing values are due to errors in data collection. First all missing values were replaced using KNN imputation and a logistic regression model was fitted with k-fold cross validation, and it resulted in misclassification error of 0.435. Afterwards, the same model was fitted with the sampled data after dropping row 54, and the error decreased to 0.388. From this experiment, it was decided to drop row 54 with 73 missing values.

Figure 2.1 shows boxplot for this column vs Class. It can be observed that there are outliers in this column, therefore, first the missing value was imputed with median and the above-mentioned experiment with logistic regression was run and got misclassification error 0.38. Again, the same model was fitted after imputation using KNN imputation, which reduced the error to 0.35. Hence, missing value was replaced using KNN imputation method.

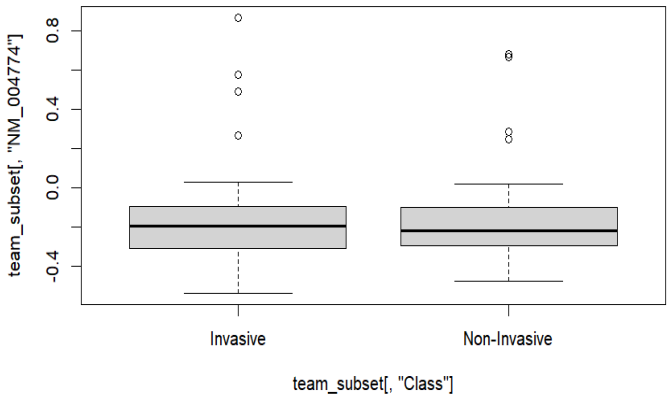

Figure 2.1: Boxplot of "NM_004774" vs "Class"

2.2. Data Scaling

The gene expression values in the dataset are in range 2 to -2. So, no scaling method was implemented.

2.3. Class imbalance

Plotting bar for both classes of cancer, there are 33 observations belonging to type 1 (invasive) and 44 observations belonging to type 2 (non-invasive). This ratio is approximately 42:58, which can be considered a minor imbalance, and hence class imbalance should not hurt supervised models for this dataset. Figure A.1 in appendix shows the bar chart for both Classes.

3. Analysis

3.1. Question 1:

3.1.1 Supervised Dimensionality Reduction: Feature selection using Two sample t-tests.

A t-test is an inferential statistical method used to determine whether there is any significant difference between the mean of two groups in a given numerical variable. Formally defined as a statistical hypothesis test used to compare the means of two population groups [4]. The two-sample t-test assumes that the data is normally distributed, and the variances are equal. It has many advantages such as the easier interpretation of the results, it

identifies the discriminant features, and it is a simple and straightforward test. Significance value (alfa) is the threshold at which we check whether the difference in means between the two groups is statistically significant. The threshold for p-value is considered as 0.05 in conventional statistics.

A function was defined to perform the t-test on each gene which returns an array of p-values. Then the names of the genes and their p-value were extracted from the array and created a data frame. By considering the significant value of p as 0.05, genes were filtered from subset data. After performing the t-test on gene sunset data, the dimension of the dataset has reduced from 2000 to 346 features. Using these reduced features subset, we can train the machine learning models to predict the outcome and evaluate them using standard metrics.

3.1.2. Unsupervised Dimensionality Reduction:

3.1.2.1. Feature Selection using Variance.

There are many methods to perform unsupervised dimensionality reduction such as PCA, t-SNE, dimension reduction using Variance, etc. Initially it was decided to do PCA to extract the significant components from the data. But the columns in the data are more than rows, hence, it is not possible to perform PCA on genes. So, it was then decided to reduce dimensions using variance because it is a simple and intuitive method. In this method, it is assumed that the features with low variance will contribute less to the overall variability of the dataset [5]. Therefore, the features with high variance were selected as they have most of the information.

Advantages of using variance to perform feature extraction are: it simplicity, compresses data, and could prevents overfitting. It is one of the fastest ways to perform dimension reduction by retaining important information in the data.

3.1.2.2. LASSO regression method.

LASSO regression stands for Least Absolute Shrinkage and Selection Operator. It is a regression analysis method used in regularization that penalizes the weights of the variables with L1 regularization. Unlike Ridge, which uses L2 regularization, the LASSO penalty can

reduce some weights to very close to 0 or even 0, avoiding overfitting and removing collinearity. The variables with higher penalties become irrelevant for the model, allowing LASSO to be used as a feature selection method.

$$L_{lasso}(\hat{\beta}) = \sum(y_i - x'_i\hat{\beta})^2 + \lambda\sum|\hat{\beta}_j|$$

In figure 3.1, we see the result of the loss function for different values of the constant α after 10-fold cross-validation is carried out. The minimum value and the optimum value of lambda are stored in the lambda_min and lambda_1se variables.

The accuracy of the logistic regression increased from 0.6428571 to 0.8571429, moving from 347 variables to a total of 29 variables after LASSO penalization. Below mentioned are the variables (genes) that resulted from the LASSO regularization, so we are going to use them to carry out further analysis.

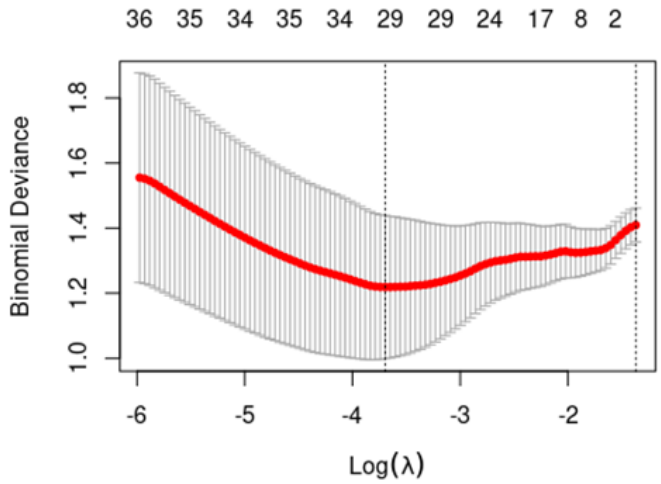

Figure 3.1: LASSO regression: Loss function for different values

3.2. Question 2:

3.2.1. Principal Component Analysis.

The correlation provides an insight into the groupings within the data frame. Here we used the 29 columns (genes) that resulted from the LASSO dimensionality reduction. We observe two distinct regions (one blue and one red). Notably, there is a high correlation of 0.74 between the genes Contig41383_RC and Contig43599_RC, suggesting that these genes are

expressed similarly across the individuals. Figure A.2 in appendix shows correlation matrix for 29 columns selected after Lasso reduction.

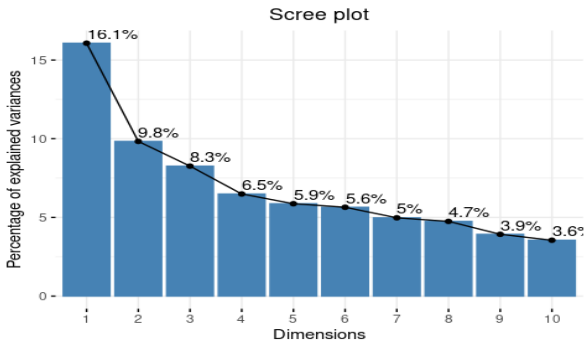

Figure 3.1: Scree Plot for first 10 principal components of genes

In table 3.1, we can see the percentage of explained variances by the first 10 components, which together account for approximately 70% of the dataset's variance. The eigenvalues are arranged from 1 to the total number of features, with the first one explaining the highest variance of the original dataset and the last one the lowest variance of the same dataset. Collectively, they sum up the entire variance. However, the initial components are typically utilized to perform the analysis instead of the complete dataset, which could be highly correlated and excessively large for some algorithms to handle effectively.

Figure A.3 in appendix, the scatterplot shows that with the first two principal components, we can already fairly accurately separate the two groups. There is still some overlap, but it is important to note that we are only using two principal components, which explain only 26% of the total dataset variance; thus, they cannot account for the entire dataset alone.

The second PCA, with the aim to reduce the number of observations rather than the number of genes was performed and its scree plot is shown in Figure A.4 in appendix. Here, the components number are 77, and we observe that the first two components already explain 31.2% of the data variability. The analysis only displays the first 10 components.

3.2.2. Hierarchical Clustering.

The dendrograms shown in Figure A.4 in appendix, explore the relationships between genes, linking pairs of nodes to other nodes or groups according to their distance. In this

instance, all distances were tested without significant differences, so the standard Euclidean distance was used to calculate the distances between genes. The Ward linkage criterion appears to separate the genes clearly into two groups, one with 7 genes and the other with 22 genes. The average and single criteria are quite similar, with almost all genes grouped in one large cluster and one gene (NM_002820) alone in another group/cluster. The complete approach also exhibits two groups; however, there is more distance between the clusters within the larger group. It is important to note that the same gene (NM_002820) forms a cluster on its own even with this linkage criterion.

The clusters in figure 3.2 demonstrate the separation of the observations based on the Canberra distance. The clusters reveal two kinds of distinct groups, with the main variation among them being the order in which the observations are linked. The single linkage criterion appears to separate the groups very poorly, which was anticipated given the overlap of the groups, as evident in the scatterplot of the first two components. Thus, the smallest distances between groups are a poor measure of closeness when such overlaps occur. Both complete and Ward linkage criteria seem to produce identical clusters. The average criterion yields a slightly different result but generally can separate the two main groups effectively.

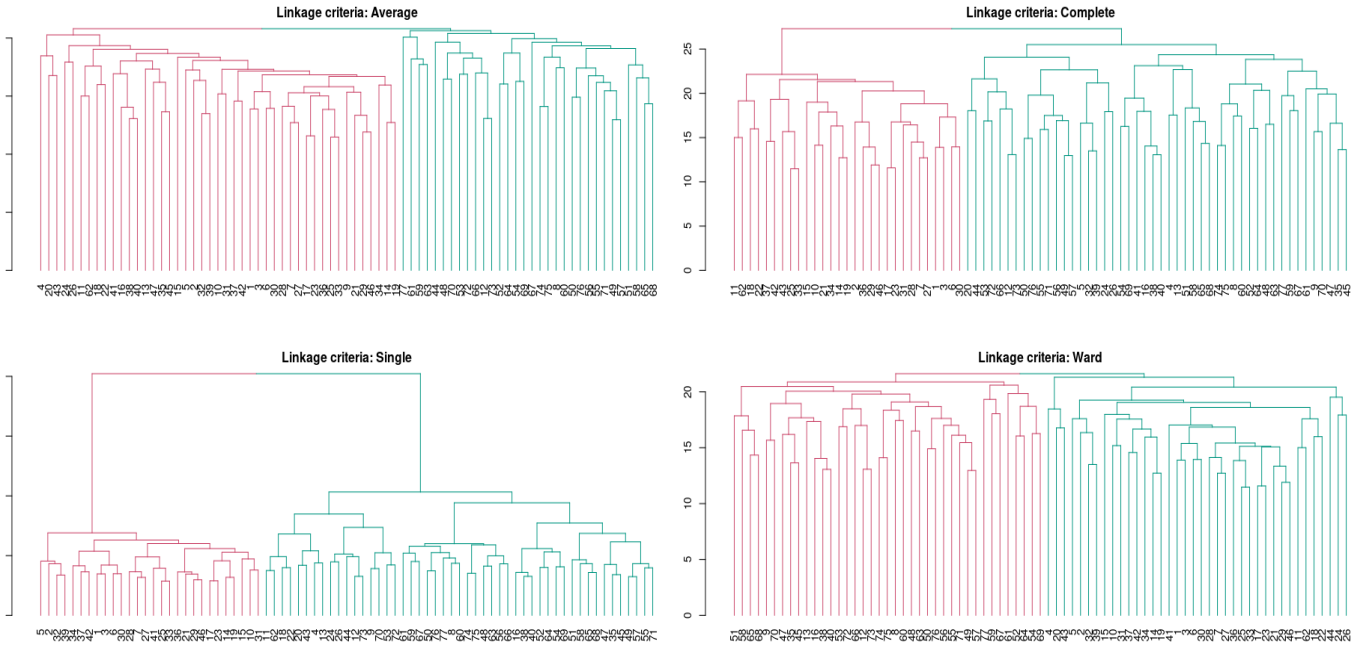

Figure 3.2: Dendrograms showing relationship between patients.

3.2.3. K-means clustering.

Figure A.6 in appendix implies that in this K-means cluster analysis, the elbow method does not provide a clear indication of whether the optimal number of clusters (k) should be 2 or 3. Therefore, we will employ Silhouette analysis. This method calculates the silhouette width, a metric that determines whether a data point is closer to its assigned group or to another group. The measure ranges from -1 to 1: a value of -1 indicates that the data point would fit better in a neighboring cluster, 0 suggests that the data point is on the boundary between two clusters, and 1 signifies that the data point is well assigned to its group.

3.2.4. Silhouette Analysis.

Using this analysis, we observe that the average silhouette width for k=3 is 0.1, whereas for k=2 it is 0.12. This indicates that with k=2, the data points are better assigned to the clusters they belong to, suggesting that two clusters provide a more cohesive and appropriate grouping of the data points compared to three clusters. Figure 3.3 shows cluster separation from silhouette analysis.

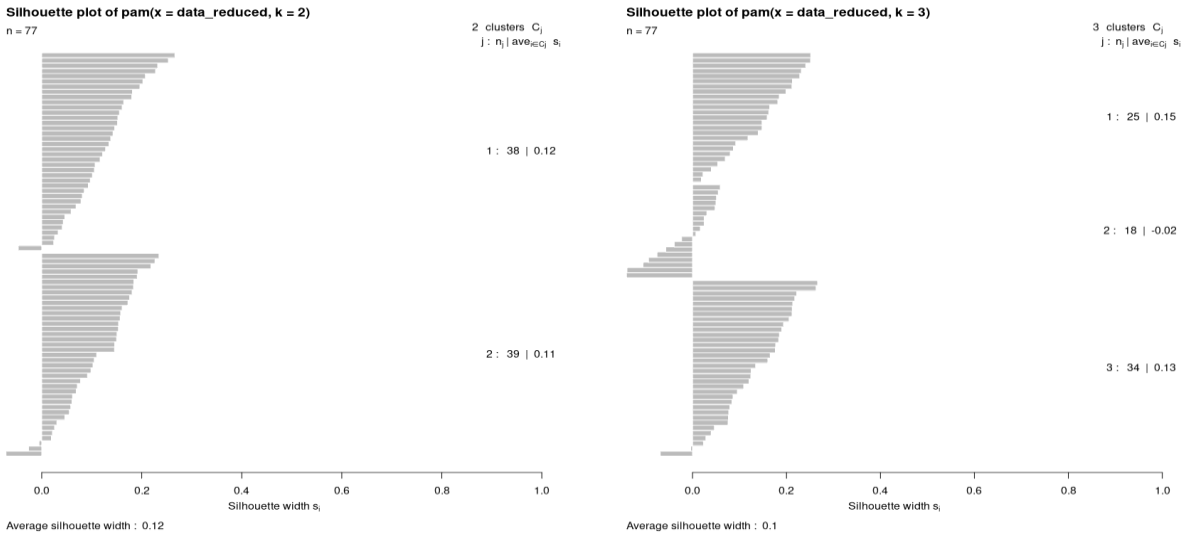

Figure 3.3: Cluster separation silhouette analysis.

3.2.5. Correlation based distance cluster.

Correlation-based distance is another approach used as a distance criterion when it comes to creating clusters, especially in the context of gene expression analysis. As demonstrated in the referenced study [6], using 1 minus the correlation between variables helps to agglomerate groups and provides a result that is congruent with previous clusters. Here, we observe two main clusters of genes: one at the top, predominantly red, featuring a gene that appears to express antagonistically with highly negative correlation, and another at the bottom, which exhibits more variability in terms of correlations compared to the first group. If we were to compare the grouping of observations to the hierarchical cluster, especially the one with complete linkage criteria, we would obtain almost identical results. This indicates that both measures, correlation-based distance, and Canberra distance, can identify similar patterns in the data grouping. Figure A.7 in appendix shows correlation-based clusters.

3.3. Question 3:

For supervised learning, Logistic regression (LR), Poisson regression (PR), LDA, KNN, random forest (RF), SVM, GLM Boost (GLMB) and XG Boost (XGB) algorithms were used. QDA could not be used as the number of observations is only 78, which is smaller compared to the number of columns, i.e. 2001. Even if data is reduced to 20 columns, QDA gives an error of rank deficiency in classes as dataset gets divided in k-fold cross validation.

Misclassification error is used as model evaluation matrix since this is a classification problem. Two types of resampling methods, namely k-fold cross validation and bootstrapping were used to get more robust errors. Experiment to estimate suitable number of k-folds was conducted for this dataset using logistic regression model. It resulted in an optimal value of k as 7, i.e. 7 folds. Figure A.8 in appendix, shows graph for number of folds vs misclassification error.

As discussed in section 3.1, multiple samples from the original dataset were used for model fitting. These samples are listed below: (1) Original dataset with 2001 gene columns. (2) Dataset filtered through two sampled t-tests in question 1 which resulted in 347 columns. (3) Considering the top 100 columns with the most variance from t-test filtered data from question 1. (4) t-test reduced columns further reduced by applying Lasso regression.

Table 1.1 shows errors of all supervised models fitted with above listed samples. Observing errors from all samples, KNN models work best on this gene expression data. The best performing model, KNN model has 0 misclassification error for Lasso reduced genes.

Model	2001 Columns		t-Test Reduced Columns		Top 100 t-Test reduced Columns		t-test + Lasso regression reduced columns	
	k-fold	Boot	k-fold	Boot	k-fold	Boot	k-fold	Boot
LR	0.388	0.518	0.363	0.47	0.55	0.481	0.155	0.243
PR	0.363	0.44	0.428	0.478	0.559	0.428	0.233	0.378
LDA	0.388	0.309	0.28	0.18	0.452	0.418	0.09	0.187
KNN	0.313	0.24	0.155	0.118	0.299	0.268	0	0
RF	0.336	0.231	0.26	0.387	0.298	0.309	0.319	0.411
SVM	0.33	0.29	0.233	0.125	0.415	0.32	0.07	0
GLMB	0.472	0.53	0.44	0.51	0.44	0.45	0.116	0.25
XGB	0.493	0.512	0.33	0.56	0.376	0.45	0.311	0.51

Table 1.1: Errors for supervised models for different samples.

The accuracy of the KNN model is somewhat surprising, as it achieves 100% accuracy, which is atypical for machine learning models, especially with biological data known for its diversity. However, there are two main reasons for this high level of performance. The first is that during the dimensionality reduction process, we performed a t-test that selected the variables where the means between the classes were different. By doing so, the performance of any distance-based algorithm would significantly improve. The second reason is the low number of observations, only 77, with 80% used for training, leaving 15 observations for testing. As a result, the accuracy levels would increase in intervals of approximately 6.67%, and while it may not always be 100%, it could be close, but not exactly 100%. Even when performing K-fold cross-validation, there aren't 100 observations for testing, so the accuracy would not be a precise integer from 1 to 100 but would instead vary between certain values.

Similar results were obtained with the SVM algorithm, where the accuracy was very high, although it did not reach 100%. Like KNN, SVM is also a distance-based algorithm for multiclass classification. In general, achieving 100% accuracy with machine learning algorithms is not entirely unusual, as even simple logistic regression algorithms have reached very high accuracy levels in our dataset.

4. Discussion

Supervised machine learning algorithms like Logistic Regression, Poison Regression, KNN, Decision Tress, XG Boost, Random Forest, GLM Boost and SVM were trained to predict invasive, or non-invasive cancer and misclassifications were validated using k-fold and bootstrapping methods. First, the whole data with 2001 columns was considered for training and KNN model generated 0.313 misclassification error. The error dropped drastically to 0.155 for k-fold and 0.118 for bootstrapping, after reducing the dimensions using two sampled t-tests. Then, the top 100 columns from reduced dataset were chosen using variance for model training and an increase in misclassification error was observed. This implies that when choosing the top 100 columns, important information is lost. Finally, genes identified from two sampled t-tests were further reduced through Lasso regression and it resulted in a perfect KNN model with 0 misclassification and SVM for the same samples came close with 0.07 misclassification error. Lasso regression for dimensionality reduction shows promising implications through this experiment, but it can be misleading as the dataset has only 78 observations.

Further analysis was done using unsupervised clustering algorithms like k-means clustering, hierarchical. It was observed that the two classes were separated fairly by using the first few components of the PCA. Clusters of features and observations were formed using hierarchical clustering and clusters of patients were observed with clear separation. To find the optimal value for number of clusters, elbow method was used and there was uncertainty to decide the optimal value of k as 2 or 3. To decide on that, Silhouette analysis was used. It was concluded that 2 was the optimal value of number of clusters.

Since the KNN model trained on Lasso reduced genes could not be further improved, genes extracted from two sample t-tests, which resulted in k-fold error of 0.155, were chosen to observe the effect of clustering on model performance. Twenty principal components from Lasso reduced genes, clusters from k-means clustering, and hierarchical clustering were used to fit KNN model with k-fold cross validation, and the misclassification error was 0.025, 0.155 and 0.142 respectively. Hence, PCA components improved the model drastically, whereas clusters from hierarchical method improved the model marginally, and clusters

from k-means method showed no improvement.

5. Conclusion

To conclude, the analysis performed on the given gene expression data showed that two sampled t-tests combined with Lasso regression can be a very effective method for dimensionality reduction. Moreover, KNN and SVM can be quite accurate in classifying whether a cancer is of invasive or non-invasive type from this gene expression data. Even a simple logistic model can be stronger when used with Lasso reduction method. Since the data is small with only 78 observations, the errors may amplify with a larger dataset and more conclusions can be drawn.

References

[1] J. Taveira De Souza, A. Carlos De Francisco and D. Carla De Macedo, "Dimensionality Reduction in Gene Expression Data Sets," in IEEE Access, vol. 7, pp. 61136-61144, 2019, doi: 10.1109/ACCESS.2019.2915519

[2] Alan Wee-Chung Liew, Ngai-Fong Law, Hong Yan, Missing value imputation for gene expression data: computational techniques to recover missing data from available information, Briefings in Bioinformatics, Volume 12, Issue 5, September 2011, Pages 498–513, https://doi.org/10.1093/bib/bbq080

[3] Brazma A, Vilo J. Gene expression data analysis. FEBS Lett. 2000 Aug 25;480(1):17-24. doi: 10.1016/s0014-5793(00)01772-5. PMID: 10967323.

[4] Mishra P, Singh U, Pandey CM, Mishra P, Pandey G. Application of student's t-test, analysis of variance, and covariance. Ann Card Anaesth. 2019 Oct-Dec;22(4):407-411. doi: 10.4103/aca.ACA_94_19. PMID: 31621677; PMCID: PMC6813708.

[5] Roberts, Aedan & Catchpoole, Daniel & Kennedy, Paul. (2018). Variance-based Feature Selection for Classification of Cancer Subtypes Using Gene Expression Data.

1-8. 10.1109/IJCNN.2018.8489279.

[6] Glazko G, Mushegian A. Measuring gene expression divergence: the distance to keep. Biol Direct. 2010 Aug 6;5:51. doi: 10.1186/1745-6150-5-51. PMID: 20691088; PMCID: PMC2928186.

Appendix

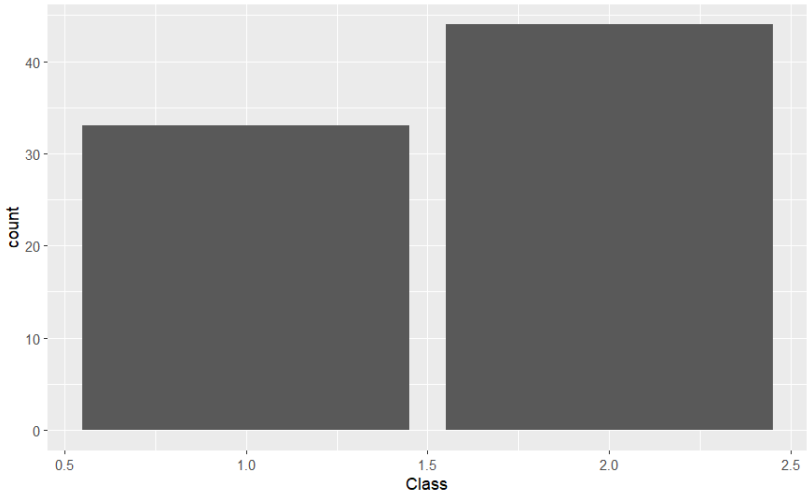

Figure A.1: Bar plot for two types of Cancer

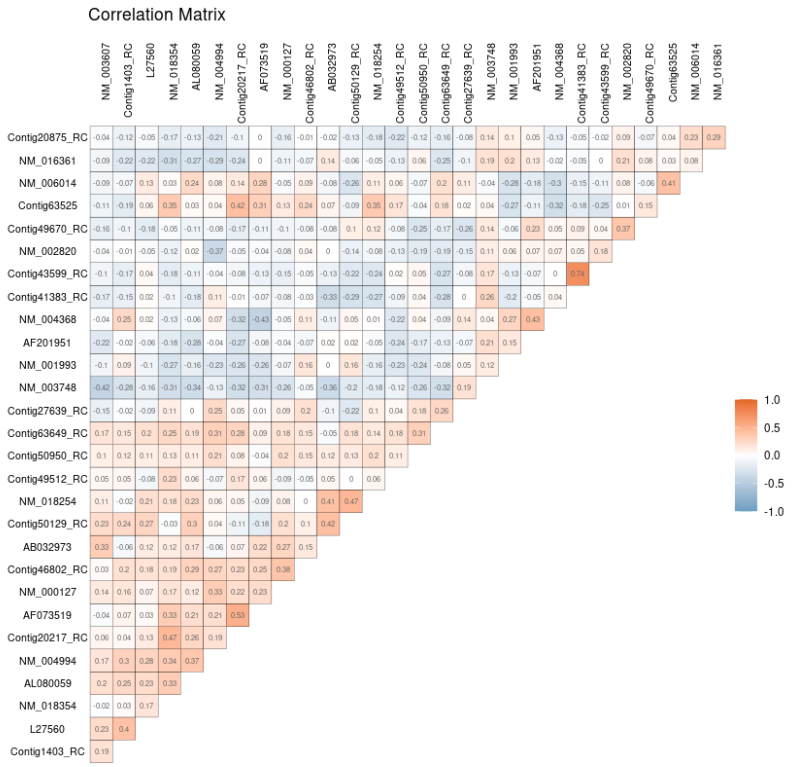

Figure A.2: Correlation matrix for 29 columns selected through Lasso reduction.

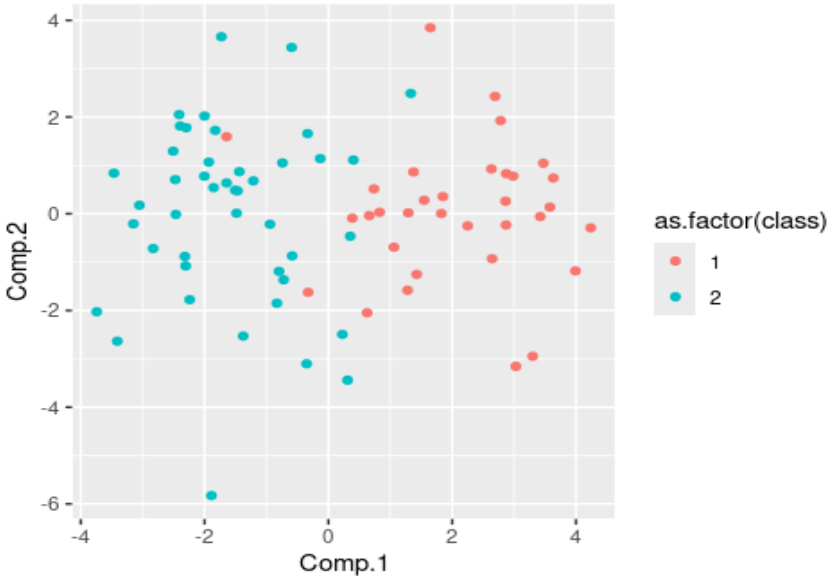

Figure A.3: Class separation by first two principal components of genes.

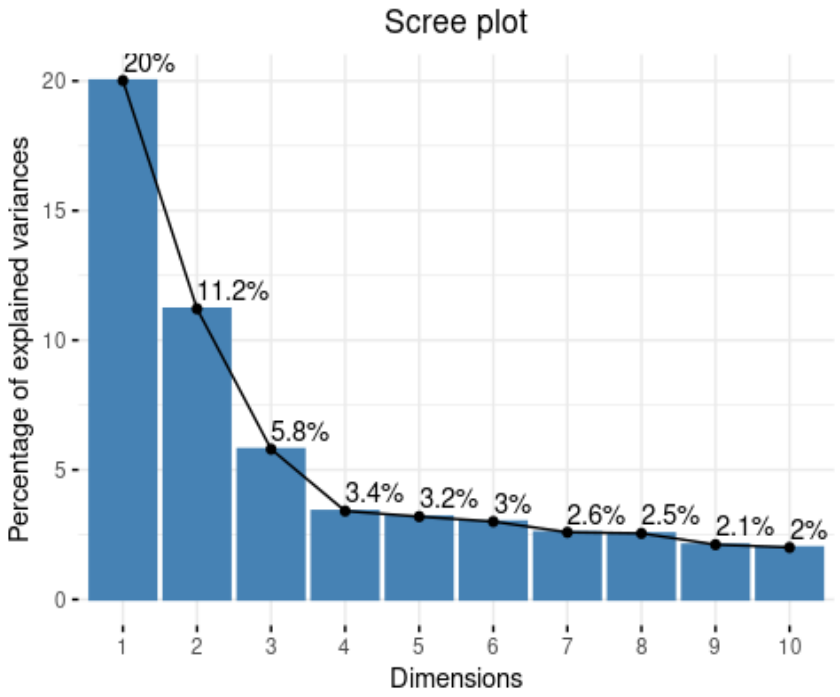

Figure A.4: First ten principal components of patients

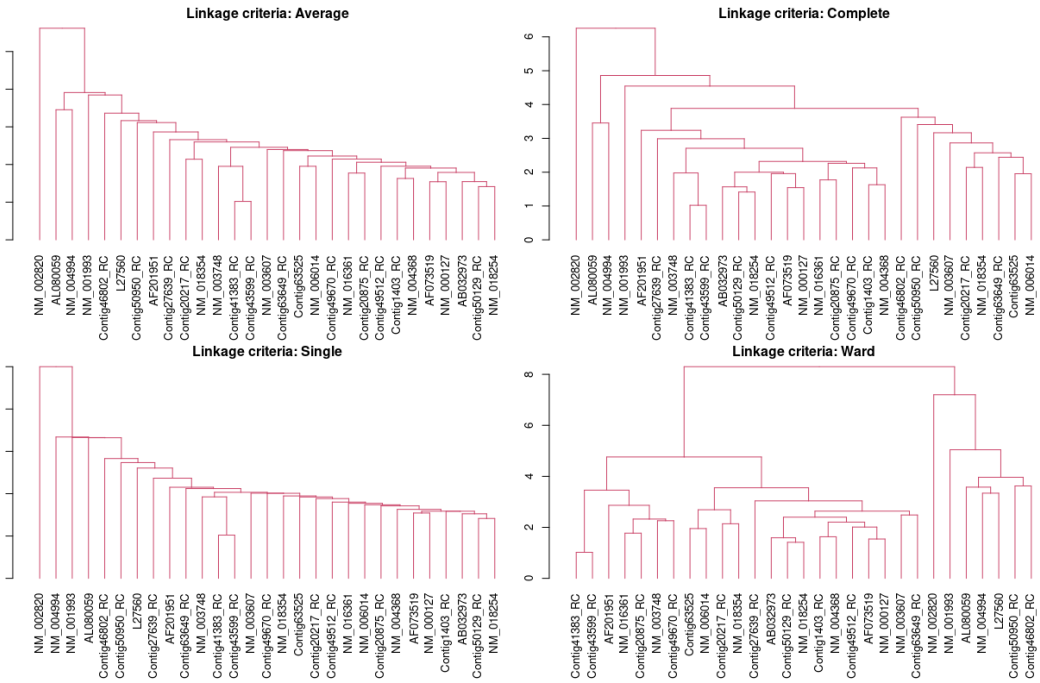

Figure A.5: Dendrograms showing relationship between genes.

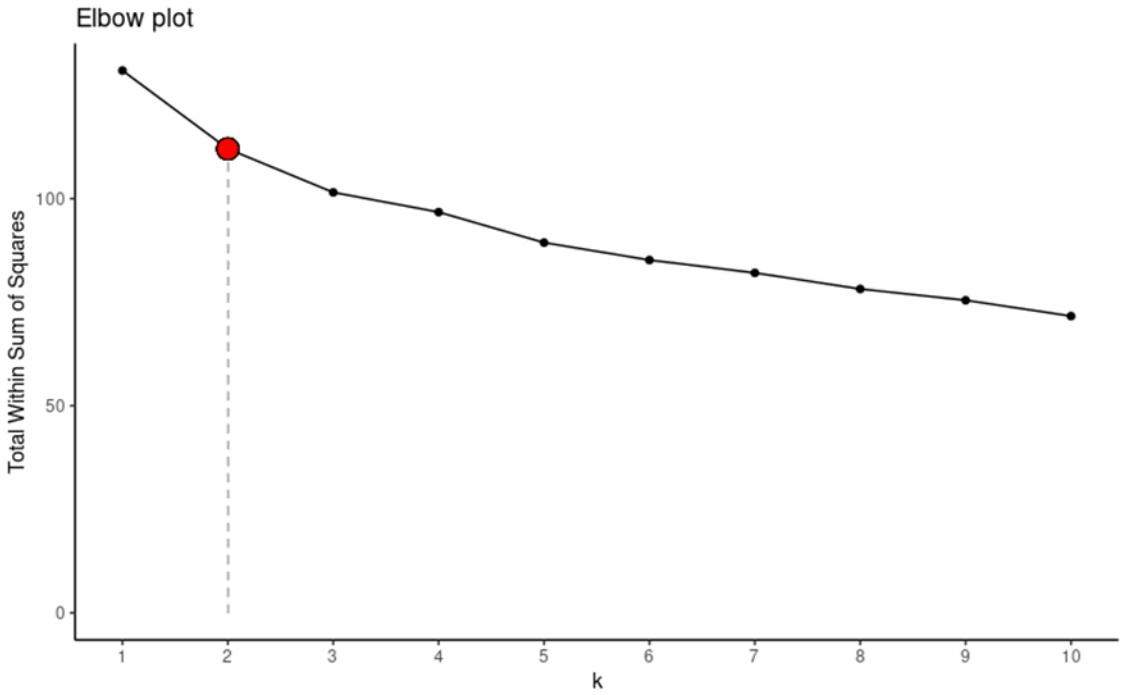

Figure A.6: Elbow graph for k-means clustering.

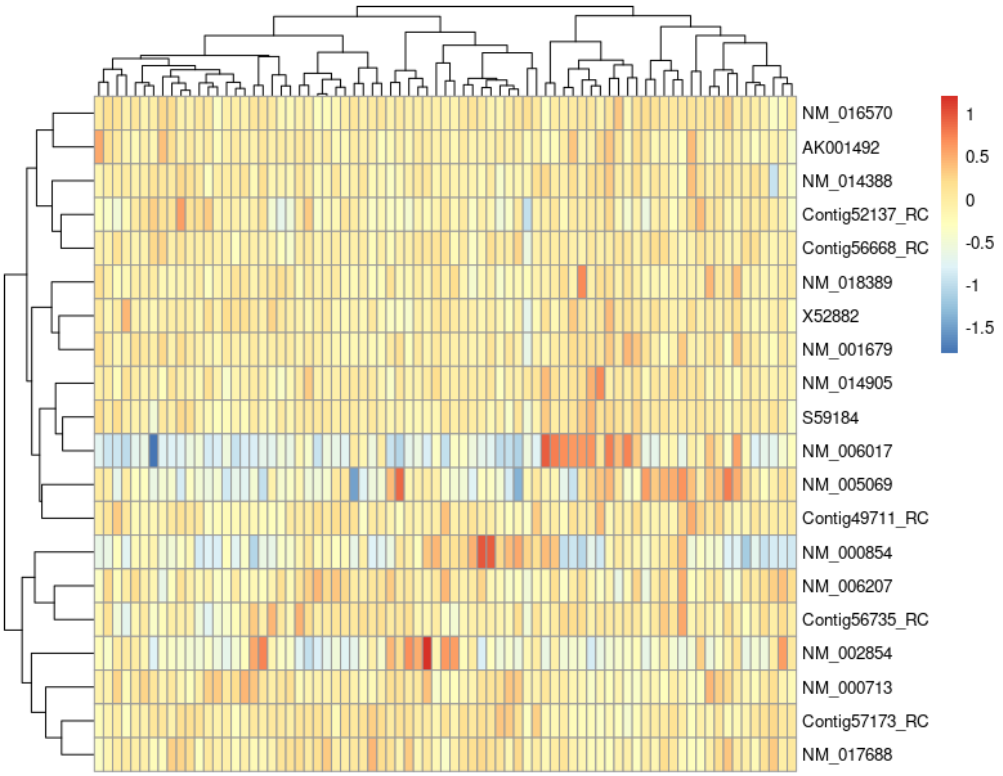

Figure A.7: Correlation-based distance clustering.

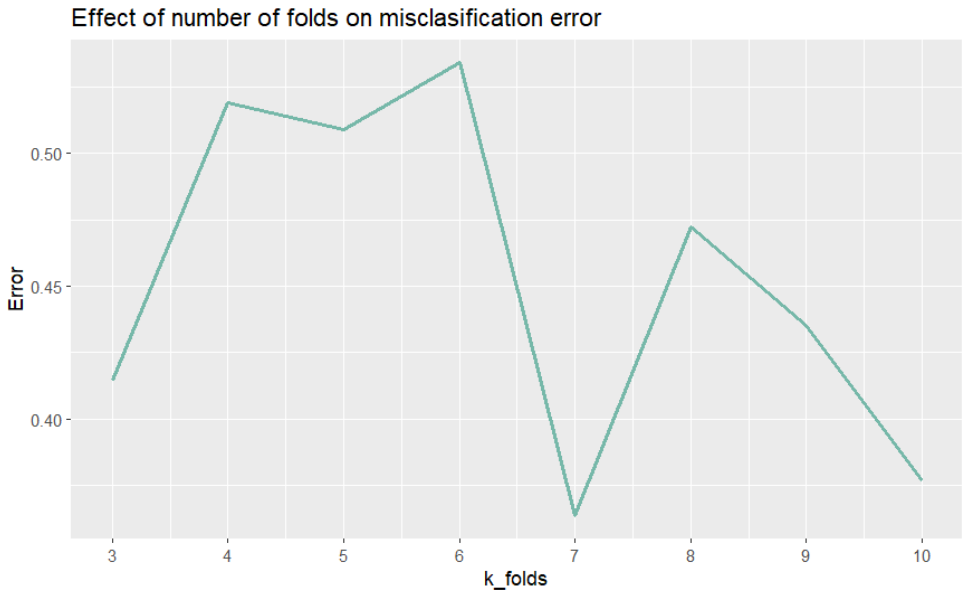

Figure A.8: Number of folds for cross validation vs Misclassification error.

R-code used for this report:

```
# Loading packages
library(dplyr) # wrangling data
library(VIM) # For imputing the data
library(purrr) # for looking through a function
library(ggplot2) # Visualizations
library(ggcorrplot)   # to create a correlation visualization
library(caret)    #for Machine learning procedures
library("DMwR") # calling library DMwR for knnimputation function
library("caTools") # package for numeric data analysis
library(class) # package for kNN
library(xgboost) # package for xgboost
library(mboost) # package for GLM Boost
library(e1071) # package for SVM
library(randomForest) # package for random forest
library(glmnet) # package for LASSO logistic regression

# load the data from csv file
raw_data <- read.csv("gene-expression-invasive-vs-noninvasive-cancer.csv")

# Data Preprocessing
# Set the seed with the highest ID number of the team
set.seed(2315880)
# Make subset from the data
subset_index <- rank(runif(1:4948))[1:2000]
team_subset <- raw_data[subset_index]
# add the variable Class
team_subset$Class <-  raw_data$Class

## Dealing with Missing Values
# get the name of the variables where there is at least 1 Missing value
var_with_NAs <- names(which(colSums(is.na(team_subset)) > 0))

# get the index of the rows where there is at least 1 Missing value
index_with_NAs <- which(rowSums(is.na(team_subset)) > 0)

# count total missing values
total_missing <- sum(is.na(team_subset))
print(paste("Total missing values: ", total_missing))
print(paste(sep = ","Percentage of the total: ", round(100 * total_missing/
(dim(team_subset)[1]*dim(team_subset)[2]),3), ' %'))
print(paste(sep = ","percentage of imcomplete columns: ",round(100 * length(var_with_NAs) /
dim(team_subset)[2],3), ' %'  ))
print(paste(sep = ", "percentage of imcomplete rows: ", round(100 * length(index_with_NAs)/
dim(team_subset)[1],3), ' %'))
```

```
# count how many missing values are there in the columns with missing values
rowSums(is.na(team_subset[index_with_NAs,]))
#dropping row 54
team_subset <- team_subset[-54, ]
dim(team_subset)
# Function to calculate misclassification error
library(MASS) # library for creating table
misclass <- function(pred, target){
  conf <- table(list(predicted=pred, observed=target)) #create confusion matrix from observed and
predicted
  sensitivity <- conf[1] / (conf[1]  + conf[2]) # calculate sensitivity
  specificity <- conf[4] / (conf[4] + conf[3]) # calculate specificity
  misclassification <- (conf[3]+conf[2])/(conf[1]+conf[2]+conf[3]+conf[4]) # calculate misclassification
  return(misclassification) # return the error
}
# Function to calculate mean from a list
mean_list <- function(list_name){
  sum <- 0
  for (i in list_name){
    if (is.na(i) == FALSE) {
 sum <- sum + i}}
  avg <- sum/length(list_name)
  return (avg)
}
#counting how many missing values
sum(is.na(team_subset))
# get the name of the variable where there is the last missing value
var_with_NAs <- names(which(colSums(is.na(team_subset)) > 0))
var_with_Nas
par(mfrow = c(1,2))
#plotting the variable to see the distribution
boxplot(team_subset[,'NM_004774'])
#plotting the variable to see the distribution by class
boxplot(team_subset[,'NM_004774']~ team_subset[,'Class'],)
# function for manual partitions for cross validation
cv_fold <- function(data_, n_folds){
  set.seed(2315880)
  data_ <- data_[sample(1:nrow(data_)),]
  if (n_folds == 3){
    dat1 <- data_[1:25, ]
    dat2 <- data_[26:50, ]
    dat3 <- data_[51:77, ]
    data_list <- list(dat1, dat2, dat3)
  }
  if (n_folds == 4){
```

```
  dat1 <- data_[1:19, ]
  dat2 <- data_[20:38, ]
  dat3 <- data_[39:57, ]
  dat4 <- data_[58:77, ]
  data_list <- list(dat1, dat2, dat3, dat4)
}
if (n_folds == 5){
  dat1 <- data_[1:15, ]
  dat2 <- data_[16:30, ]
  dat3 <- data_[31:45, ]
  dat4 <- data_[46:60, ]
  dat5 <- data_[61:77, ]
  data_list <- list(dat1, dat2, dat3, dat4, dat5)
}
if (n_folds == 6){
  dat1 <- data_[1:13, ]
  dat2 <- data_[14:26, ]
  dat3 <- data_[27:39, ]
  dat4 <- data_[40:52, ]
  dat5 <- data_[53:65, ]
  dat6 <- data_[66:77, ]
  data_list <- list(dat1, dat2, dat3, dat4, dat5, dat6)
}
if (n_folds == 7){
  dat1 <- data_[1:11, ]
  dat2 <- data_[12:22, ]
  dat3 <- data_[23:33, ]
  dat4 <- data_[34:44, ]
  dat5 <- data_[45:55, ]
  dat6 <- data_[56:66, ]
  dat7 <- data_[67:77, ]
  data_list <- list(dat1, dat2, dat3, dat4, dat5, dat6, dat7)
}
if (n_folds == 8){
  dat1 <- data_[1:9, ]
  dat2 <- data_[10:19, ]
  dat3 <- data_[20:28, ]
  dat4 <- data_[29:38, ]
  dat5 <- data_[39:47, ]
  dat6 <- data_[48:57, ]
  dat7 <- data_[58:67, ]
  dat8 <- data_[68:77, ]
  data_list <- list(dat1, dat2, dat3, dat4, dat5, dat6, dat7, dat8)
}
if (n_folds == 9){
```

```
  dat1 <- data_[1:8, ]
  dat2 <- data_[9:17, ]
  dat3 <- data_[18:25, ]
  dat4 <- data_[26:34, ]
  dat5 <- data_[35:42, ]
  dat6 <- data_[43:51, ]
  dat7 <- data_[52:59, ]
  dat8 <- data_[60:68, ]
  dat9 <- data_[69:77, ]
  data_list <- list(dat1, dat2, dat3, dat4, dat5, dat6, dat7, dat8, dat9)
 }
 if (n_folds == 10){
  dat1 <- data_[1:7, ]
  dat2 <- data_[8:14, ]
  dat3 <- data_[15:22, ]
  dat4 <- data_[23:30, ]
  dat5 <- data_[31:38, ]
  dat6 <- data_[39:46, ]
  dat7 <- data_[47:54, ]
  dat8 <- data_[55:68, ]
  dat9 <- data_[69:70, ]
  dat10 <- data_[71:78, ]
  data_list <- list(dat1, dat2, dat3, dat4, dat5, dat6, dat7, dat8, dat9, dat10)
 }
 return(data_list)}

#Question 1:
## Try median with cross validation
experiment_data <- team_subset # making a temporary dataset to run the experiment
# replace missing value using median
value_to_impute <- median(experiment_data['NM_004774'], na.rm = T)
experiment_data <- experiment_data %>%
      replace(is.na(.), value_to_impute)
# change class labels 1 and 2 to 0 and 1
experiment_data$Class <- case_when(
 experiment_data$Class == 1~0,
 experiment_data$Class == 2~1)
data_list <- cv_fold(experiment_data, 5)
# run logistic regression model for all 5 folds and an average error
avg_error <- list()
for (i in data_list){
 cv_test <- as.data.frame(i)
 cv_train <- setdiff(experiment_data, cv_test)
 LR <- glm(Class ~., data = cv_train, family = binomial)
 LR_prediction <- predict(LR, newdata = cv_test, type = 'response')
```

```
  LR_prediction <- case_when(
    LR_prediction < 0.5 ~ 0,
    TRUE ~ 1
    )
  error <- misclass(LR_prediction, cv_test$Class)
  avg_error <- append(avg_error, error)
}
mean_list(avg_error) # get mean error from all 6 folds

## Try KNN impute method
# run an experiment of knn imputation for all k between 1 and 10, and check which k gives least error
kn <- seq(1,10, by = 1)
error_list <- list()
impute_number <- list()
experiment_data <- team_subset
for (impute in kn){
  # impute missing value using kNNimpute
  experiment_data <- knnImputation(data = experiment_data, k = impute, scale = TRUE, distData =
NULL)
  # Logistic Regression
  data_list <- cv_fold(experiment_data, 5)
  avg_error <- list()
  for (i in data_list){
    cv_test <- as.data.frame(i)
    cv_train <- setdiff(experiment_data, cv_test)
    LDA <- lda(Class ~.  , data=cv_train) # perform LDA on subset
    lda_prediction <- predict(LDA, newdata = cv_test)
    error <- misclass(lda_prediction$class, cv_test$Class)
    avg_error <- append(avg_error, error)}
  err <- mean_list(avg_error)
  impute_number <- append(impute_number, impute)
  error_list <- append(error_list, err)
}
print(min(unlist(error_list)))
impute_df = data.frame(unlist(impute_number),unlist(error_list))
names(impute_df) = c("k_value","Error")
ggplot(impute_df, aes(x=k_value, y=Error)) +
  geom_line( color="#69b3a2", linewidth=1, alpha=0.9, linetype=1) +
  scale_x_continuous(breaks= kn) +
  ggtitle("Effect of no. of neighbours on error")
#Finding the index of the gene which contain  a null value
indi <- which(is.na(team_subset[,'NM_004774']))
team_subset[,'NM_004774'][indi]
# Experiment to check imputed values for different k in kNN impute
kn <- seq(1,10, by = 1)
```

```
imputed <- list()
experiment_data <- team_subset
indi <- which(is.na(experiment_data['NM_004774']))
for (impute in kn){
  # impute missing value using kNNimpute
  experiment_data <- knnImputation(data = experiment_data, k = impute, scale = TRUE, distData =
NULL)
  imputed <- append(imputed, experiment_data['NM_004774'][indi])}
print(imputed)

# Check class imbalance
team_subset['NM_004774']
# Impute missing values using kNN
new_df <- knnImputation(data = team_subset, k = 2, scale = TRUE, distData = NULL)
# Change classes from 1 and 2 to 0 and 1 for using in logistic regression
new_df$Class <- case_when(
  new_df$Class == 1~0,
  new_df$Class == 2~1)
# plot bar plot to check class imbalance
library("ggplot2")
ggplot(data = new_df, aes(x=Class))+ geom_bar()

# T-test
# Function for Two sample t-test
gene_func <- function(gene) {
  results <- t.test(new_df[[gene]] ~ new_df$Class)
  return(results$p.value)
}
# Applying t-test function
p_values <- sapply(names(new_df)[-ncol(new_df)], gene_func)
# Storing t-test results in a data frame
t_test_results_df <- data.frame(Genes = names(p_values), P_Value = p_values)
# Filtering genes
filtered_genes <- t_test_results_df[t_test_results_df$P_Value < 0.05, ]
# Final subset after performing dimensionality reduction using two sample t-test
filtered_df <- new_df[colnames(t(filtered_genes))]
filtered_df$Class <- new_df$Class
dim(filtered_df)

# Using variance to perform dimensionality reduction
# gene_variance <- apply(new_df, 2, var)
# filtered_cols <- names(gene_variance[gene_variance > 0.252])
# #choosing first 50 samples with highest variance
# filtered_var_df <- new_df[, filtered_cols]
# dim(filtered_var_df)
```

```
# Perform PCA
# len <- length(filtered_var_df)
# pca_ <- princomp(filtered_var_df[,1:len-1], cor = TRUE) # create principal components
# summary(pca_, loadings = FALSE, cutoff=.1) # summarize created principal components
# plot(pca_) # plot principal components
# Make dataset from top 25 PCA components
# pc <- pca_$score[,1:25]
# filtered_var_df <- as.data.frame(pc)
# dim(filtered_var_df)

#Question 2:
# LASSO Logistic regression regularisation
# change the name of the Class for avoiding confusion when performing the logistic regression
filtered_ <- filtered_df
filtered_$Class <- ifelse(filtered_$Class == 1 , 'non-inv' , 'inv')
# Separating the data into training and testing
set.seed(2315880)
index_train <- filtered_$Class %>%
      createDataPartition(p = 0.8, list = F)
# Creating the training and testing datasets
train.data <- filtered_[index_train,]
test.data <- filtered_[-index_train,]

# Logistic regression with all 347 variables
GLM_model <- glm(as.factor(Class)~. ,data = train.data, family = binomial)
# predicting with the model we just fit
class_num = which(names(test.data)== 'Class')
glm_prob <- predict.glm(GLM_model, test.data[,-class_num], type= 'response')
contrasts(as.factor(train.data$Class))
glm_predict <- rep('inv',nrow(test.data))
glm_predict[glm_prob>.5] <- 'non-inv'
#confusion matrix
table(pred= glm_predict, true=test.data$Class)
#accuracy
mean(glm_predict==test.data$Class)

# Creating a matrix
x <- model.matrix(Class~. , train.data)
# Preparing the dependent variable for the model
y <- ifelse(train.data$Class=='inv', 1, 0)
set.seed(2315880)
# Fitting the model
cv.lasso <- cv.glmnet(x, y, alpha = 1, family = "binomial") #alpha =1 for LASSO
plot(cv.lasso)
# min value of lambda
```

```
lambda_min <- cv.lasso$lambda.min
# best value of lambda (minimun number of variables)
lambda_1se <- cv.lasso$lambda.1se
# prepare the data into a matrix format
test.data_matrix <- model.matrix(Class~.,test.data)
#predict class with the s value equal to lambda_min
lasso_prob <- predict(cv.lasso, newx = test.data_matrix, s = lambda_min, type= 'response')
#translate probabilities to predictions
lasso_predict <- rep('non-inv',nrow(test.data))
lasso_predict[lasso_prob>.5] <- 'inv'
#confusion matrix
table(pred=lasso_predict,true=test.data$Class)
#accuracy
mean(lasso_predict==test.data$Class)
#regression coefficients
coefficients <- as.matrix(coef(cv.lasso, s = lambda_min))
variables_names <- names(coefficients[coefficients!=0,])
variables_names <-variables_names[-(variables_names == "(Intercept)") ]
variables_names
data_reduced <- new_df[,variables_names]
data_reduced_with_class <- data_reduced %>% mutate(Class = new_df$Class)

#Correlation
library(correlation)
# Create the correlation matrix sorted for easier visualization.
corr_matrix <- cor_sort(cor(data_reduced),distance = "correlation")
#plot the correlation matrix
ggcorrplot(corr_matrix,
          type = "upper",
          outline.color = "black",
          ggtheme = ggplot2::theme_void(),
          colors = c("#6D9EC1", "white", "#E46726"),
          lab = TRUE,
          lab_size = 2,
          lab_col = "grey40",
          title = 'Correlation Matrix\n' ,
          legend.title = "",
          tl.cex = 8,
          tl.col = 45,
          tl.srt = 90)+
        scale_x_discrete(position='top')

# PCA (features)
#fitting the data into a PCA
pca.fit <- princomp(data_reduced, cor = TRUE)
```

```
summary(pca.fit)
plot(pca.fit)
data_pca <- pca.fit$scores[,1:10]
#names(data) <- c('PC 1', 'PC 2')
data_pca <- as.data.frame(data_pca)
data_pca$class <- new_df$Class
ggplot( data_pca, aes(x = Comp.1, y = Comp.2, colour = as.factor(class)))+
  geom_point()

## PCA (observations)
team_subset.t <- t(new_df[,-1])
#fitting the data into a PCA
pca.t.fit <- princomp(team_subset.t, cor = TRUE)
summary(pca.t.fit)
plot(pca.t.fit)

# Hierarchical Clustering (genes)
library(dendextend)
# distance matrix
distance_matrix <- dist(t(data_reduced),method = 'euclidean')
# "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski".
h_cluster_average <- hclust(distance_matrix, method = 'average')
h_cluster_average_d <- as.dendrogram(h_cluster_average)
h_cluster_average_colour <- color_branches(h_cluster_average_d, k=3)
h_cluster_complete <- hclust(distance_matrix, method = 'complete')
h_cluster_complete_d <- as.dendrogram(h_cluster_complete)
h_cluster_complete_colour <- color_branches(h_cluster_complete_d, k=3)
h_cluster_single <- hclust(distance_matrix, method = 'single')
h_cluster_single_d <- as.dendrogram(h_cluster_single)
h_cluster_single_colour <- color_branches(h_cluster_single_d, k=3)
h_cluster_centroid <- hclust(distance_matrix, method = 'mcquitty')
h_cluster_centroid_d <- as.dendrogram(h_cluster_centroid)
h_cluster_centroid_colour <- color_branches(h_cluster_centroid_d, k=3)
#Plotting the clusters
par(mfrow = c(2,2),mar = c(2, 2, 2, 2))
plot(h_cluster_average_colour, main="Linkage criteria: Average", cex = 0.6)
plot(h_cluster_complete_colour, main="Linkage criteria: Complete", cex = 0.6)
plot(h_cluster_single_colour, main="Linkage criteria: Single", cex = 0.6)
plot(h_cluster_centroid_colour, main = "Linkage criteria: Mcquitty", cex = 0.6)
# Herarchical cluster (Observations)
library(dendextend)
# distance matrix
distance_matrix <- dist(data_reduced, method = "canberra" )
# "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski".
h_cluster_average <- hclust(distance_matrix, method = 'average')
```

```
h_cluster_average_d <- as.dendrogram(h_cluster_average)
h_cluster_average_colour <- color_branches(h_cluster_average_d, k=2)
h_cluster_complete <- hclust(distance_matrix, method = 'complete')
h_cluster_complete_d <- as.dendrogram(h_cluster_complete)
h_cluster_complete_colour <- color_branches(h_cluster_complete_d, k=2)
h_cluster_single <- hclust(distance_matrix, method = 'single')
h_cluster_single_d <- as.dendrogram(h_cluster_single)
h_cluster_single_colour <- color_branches(h_cluster_single_d, k=2)
h_cluster_centroid <- hclust(distance_matrix, method = 'mcquitty')
h_cluster_centroid_d <- as.dendrogram(h_cluster_centroid)
h_cluster_centroid_colour <- color_branches(h_cluster_centroid_d, k=2)
#plotting the clusters
par(mfrow = c(2,2),mar = c(2, 2, 2, 2))
plot(h_cluster_average_colour, main="Linkage criteria: Average", cex = 0.6)
plot(h_cluster_complete_colour, main="Linkage criteria: Complete", cex = 0.6)
plot(h_cluster_single_colour, main="Linkage criteria: Single", cex = 0.6)
plot(h_cluster_centroid_colour , main = "Linkage criteria: Mcquitty", cex = 0.6)

# Correlation-based distance Cluster (genes and observations)
#Using 1-cor - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2928186/
# Pairwise correlation between samples (columns)
cols.cor <- cor(t(team_subset[,1:20]), use = "pairwise.complete.obs", method = "pearson")
# Pairwise correlation between rows (genes)
rows.cor <- cor(team_subset[,1:20], use = "pairwise.complete.obs", method = "pearson")

# Plot the heatmap
library(pheatmap)
pheatmap(team_subset[,1:20], scale = "row",
  clustering_distance_cols = as.dist(1 - rows.cor),
  clustering_distance_rows = as.dist(1 - cols.cor))

#  K means clustering
# K-means
model <- kmeans(data_reduced, centers = 2)
# Elbow method - choosing the correct K
tot_withinss <- map_dbl(1:10, function(k){
      model <- kmeans(x = data_reduced, centers = k)
      model$tot.withinss
})
elbow_df <- data.frame(
      k = 1:10,
      tot_withinss = tot_withinss
)

# Elbow plot
```

```
ggplot(elbow_df, aes(x = k, y = tot_withinss)) + geom_line() + geom_point()+
            geom_segment(aes(x = k[-1][which.max(tot_withinss[-1])],
            xend = k[-1][which.max(tot_withinss[-1])], y =  0,
            yend =  max(tot_withinss[-1])), color="grey",
            linetype="dashed")+ geom_point(aes(x = k[2], y = tot_withinss[2]),
            fill = 'red', shape = 21 ,colour = 'black', size = 5)+ labs(title = 'Elbow plot',
            x = 'k', y = 'Total Within Sum of Squares')+
            scale_x_continuous(breaks = 1:10)+theme_classic()

# Silhouette analysis
library(cluster)
# plotting the Average silhoutte width for k = 2
pam_k2 <- pam(data_reduced, k = 2)
sil2_plot <- silhouette(pam_k2)
# plotting the Average silhoutte width for k = 3
pam_k3 <- pam(data_reduced, k = 3)
sil3_plot <- silhouette(pam_k3)
par(mfrow = c(1,2))
plot(sil2_plot)
plot(sil3_plot)
#using purrr::map_bdl to iterate and calculate the sil_width over and over
library(purrr)
sil_width <- map_dbl(2:10, function(k){
       model <- pam(x = data_reduced, k = k)
       model$silinfo$avg.width})
sil_df <- data.frame(
       k = 2:10,
       sil_width = sil_width)
# plotting the Average silhoutte width
ggplot(sil_df, aes(x = k, y = sil_width)) +
  geom_line() +
    geom_point()+
  scale_x_continuous(breaks = 2:max(sil_df))+
  labs(title = 'Optimal number of Clusters',
      y ='Average silhoutte width',
      x = 'k')+
  geom_segment(aes(x = k[which.max(sil_width)],
                xend = k[which.max(sil_width)],
                y =  0,
                yend =  max(sil_width)),color="grey",linetype="dashed")+
  geom_point(aes(x = k[which.max(sil_width)],
               y = max(sil_width)),
           fill = 'red',
           shape = 21 ,
           colour = 'black',
```

```
          size = 5)+
  theme_classic()
```

```
# Question 3:
# Use lasso reduced data
lasso_reduced <- data_reduced
lasso_reduced$Class <- new_df$Class
lasso_reduced
#Adding class column to filtered data
#filtered_var_df <- filtered_df
filtered_var_df <- lasso_reduced
#filtered_var_df$Class <- new_df$Class
# shuffle the dataframe by rows
set.seed(2315880)
filtered_var_df= filtered_var_df[sample(1:nrow(filtered_var_df)), ]
# make train and test subsets
sample = sample.split(filtered_var_df$Class, SplitRatio = 0.8)
train = subset(filtered_var_df, sample == TRUE)
test  = subset(filtered_var_df, sample == FALSE)
print(dim(train))
print(dim(test))

# Logistic Regression
ks <- seq(3,10, by = 1)
k_numbers <- list()
errors <- list()
for (each in ks){
  data_list <- cv_fold(filtered_var_df, each)
  # Logistic Regression with k-fold
  avg_error <- list()
  # run logistic regressions for all 6 folds
  for (i in data_list){
    cv_test <- as.data.frame(i) # make test data from selected fold
    cv_train <- setdiff(filtered_var_df, cv_test) # make train data with rows other than test
    LR <- glm(Class ~., data = cv_train, family = binomial)
    LR_prediction <- predict(LR, newdata = cv_test, type = 'response')
    # change probabilities to labels
    LR_prediction <- case_when(
      LR_prediction < 0.5 ~ 0,
      TRUE ~ 1
```

```
    )
   error <- misclass(LR_prediction, cv_test$Class)
   avg_error <- append(avg_error, error)
  }
  mean_err <- mean_list(avg_error)
  errors <- append(errors, mean_err)
  k_numbers <- append(k_numbers, each)
}
print(min(unlist(errors)))
err_df = data.frame(unlist(k_numbers),unlist(errors))
names(err_df) = c("k_folds","Error")
ggplot(err_df, aes(x=k_folds, y=Error)) +
  geom_line( color="#69b3a2", linewidth=1, alpha=0.9, linetype=1) +
  scale_x_continuous(breaks= kn) +
  ggtitle("Effect of number of folds on misclasification error")

# Logistic Regression with bootstrapping
library(boot)
# function for logistic regression for bootstrapping
LR_function <- function(formula, data, indices){
  d <- data[indices, ]
  LR <- glm(formula, data = d, family = binomial(link = logit))
  LR_prediction <- predict(LR, newdata = test, type = 'response')
  LR_prediction <- case_when(
    LR_prediction < 0.5 ~ 0,
    TRUE ~ 1
    )
  return(misclass(LR_prediction, test$Class))
}
# repeat logistic regression for 20 bootstraps
LR_reps <- boot(data=train, statistic=LR_function, R=20, formula=Class ~.)
mean_list(LR_reps$t)

# Poisson Regression
# Poisson Regression with k-fold
data_list <- cv_fold(filtered_var_df, 7)
avg_error <- list()
for (i in data_list){
  cv_test <- as.data.frame(i)
  cv_train = setdiff(filtered_var_df, cv_test)
  PR <- glm(Class ~., data = cv_train, family = poisson(link = log))
  PR_prediction <- predict(PR, newdata = cv_test, type = 'response')
  PR_prediction <- case_when(
    PR_prediction < 0.5 ~ 0,
    TRUE ~ 1)
```

```
  error <- misclass(PR_prediction, cv_test$Class)
  avg_error <- append(avg_error, error)
}
mean_list(avg_error)

# Poisson Regression with bootstrapping
# function for poisson regression for bootstrapping
PR_function <- function(formula, data, indices){
  d <- data[indices, ]
  PR <- glm(formula, data = d, family = poisson(link = log))
  PR_prediction <- predict(PR, newdata = test, type = 'response')
  PR_prediction <- case_when(
    PR_prediction < 0.5 ~ 0,
    TRUE ~ 1)
  return(misclass(PR_prediction, test$Class))
}
# repeating poisson regression for 20 bootstraps
PR_reps <- boot(data=train, statistic=PR_function, R=20, formula=Class ~.)
mean_list(PR_reps$t)

# LDA
# LDA with k-fold
avg_error <- list()
for (i in data_list){
  cv_test <- as.data.frame(i)
  cv_train = setdiff(filtered_var_df, cv_test)
  LDA <- lda(Class ~.  , data=cv_train) # perform LDA on subset
  lda_prediction <- predict(LDA, newdata = cv_test)
  error <- misclass(lda_prediction$class, cv_test$Class)
  avg_error <- append(avg_error, error)
}
mean_list(avg_error)

# LDA with bootstrapping
LDA_function <- function(formula, data, indices){
  d <- data[indices, ]
  LDA <- lda(formula , data=d)
  lda_prediction <- predict(LDA, newdata = test)
  return(misclass(lda_prediction$class, test$Class))
}
LDA_reps <- boot(data=train, statistic=LDA_function, R=20, formula=Class ~.)
mean_list(LDA_reps$t)

# QDA with k-fold
 # avg_error <- list()
```

```
 # for (i in data_list){
 #   cv_test <- as.data.frame(i)
 #   cv_train = setdiff(filtered_var_df, cv_test)
 #   QDA <- qda(Class ~.  , data=cv_train) # perform LDA on subset
 #   qda_prediction <- predict(QDA, newdata = cv_test)
 #   error <- misclass(qda_prediction$class, cv_test$Class)
 #   avg_error <- append(avg_error, error) }
 # mean_list(avg_error)


# QDA with bootstrapping
# QDA_function <- function(formula, data, indices){
#   d <- data[indices, ]
#   QDA <- qda(formula , data=d)
#   qda_prediction <- predict(QDA, newdata = test)
#   return(misclass(qda_prediction$class, test$Class))}
# QDA_reps <- boot(data=train, statistic=QDA_function, R=100, formula=Class ~.)
# mean_list(QDA_reps$t)


# kNN
# KNN with k-fold
k = seq(1, 20, by=1)
k_value <- list()
cv_error <- list()
for (val in k)
{
  avg_error <- list()
  for (i in data_list){
    cv_test <- as.data.frame(i)
    cv_train = setdiff(filtered_var_df, cv_test)
    classifier_knn <- knn(train = cv_train,
                          test = cv_test,
                          cl = cv_train$Class,
                          k = val)
    error <- misclass(classifier_knn, cv_test$Class)
    avg_error <- append(avg_error, error)
  }
  k_value <- append(k_value, val)
  e <- mean_list(avg_error)
  cv_error <- append(cv_error, e)
}
print(min(unlist(cv_error)))
knn_score_df = data.frame(unlist(k_value),unlist(cv_error))
names(knn_score_df) = c("k","cv_error")
```

```
#Plot
ggplot(knn_score_df, aes(x=k, y=cv_error)) +
  geom_line( color="#69b3a2", linewidth=1, alpha=0.9, linetype=1) +
  scale_x_continuous(breaks= k) +
  ggtitle("Effect of no. of neighbours on error")

# KNN with bootstrapping
knn_function <- function(data, indices){
  d <- data[indices, ]
  classifier_knn <- knn(train = d,
                        test = test,
                        cl = d$Class,
                        k = val)

  return(misclass(classifier_knn, test$Class))
}
k = seq(1,20, by=1)
k_value <- list()
cv_error <- list()
for (val in k)
{
  KNN_reps <- boot(data=train, statistic=knn_function, R=20)
  ee <- mean_list(KNN_reps$t)
  k_value <- append(k_value, val)
  cv_error <- append(cv_error, ee)
  }
print(min(unlist(cv_error)))
knn_score_df = data.frame(unlist(k_value),unlist(cv_error))
names(knn_score_df) = c("k","cv_error")
ggplot(knn_score_df, aes(x=k, y=cv_error)) +
  geom_line( color="#69b3a2", linewidth=1, alpha=0.9, linetype=1) +
  scale_x_continuous(breaks= k) +
  ggtitle("Effect of no. of neighbours on error")


# Random Forest
# Random forest with k-fold
trees = seq(1,20, by = 1)
n_trees <- list()
rf_error <- list()
for (tree in trees)
{
  avg_error <- list()
  for (i in data_list){
    cv_test <- as.data.frame(i)
```

```
    cv_train = setdiff(filtered_var_df, cv_test)
    rf <- randomForest(Class~., data=cv_train, proximity=TRUE, ntree = tree, set.seed(2315880))
    rf_prediction <- predict(rf, newdata = cv_test)
    # rf_prediction <- case_when(
    #   rf_prediction < 0.5 ~ 0,
    #   TRUE ~ 1)
    error <- misclass(rf_prediction, cv_test$Class)
    avg_error <- append(avg_error, error)
  }
  n_trees <- append(n_trees, tree)
  e <- mean_list(avg_error)
  rf_error <- append(rf_error, e)
}
print(min(unlist(rf_error)))
rf_score_df = data.frame(unlist(n_trees),unlist(rf_error))
names(rf_score_df) = c("n_trees","Error")
ggplot(rf_score_df, aes(x=n_trees, y=Error)) +
  geom_line(color="#69b3a2", linewidth=1, alpha=0.9, linetype=1) +
  scale_x_continuous(breaks= trees) +
  ggtitle("No. of trees vs error")

# Random forest with bootstraping
RF_function <- function(formula, data, indices){
  d <- data[indices, ]
  rf <- randomForest(formula, data=d, proximity=TRUE, ntree = tree, set.seed(2315880))
  rf_prediction <- predict(rf, newdata = test)
  # rf_prediction <- case_when(
  #   rf_prediction < 0.5 ~ 0,
  #   TRUE ~ 1)
  return(misclass(rf_prediction, test$Class))
}
trees = seq(1,20, by = 1)
n_trees <- list()
rf_error <- list()
for (tree in trees){
  RF_reps <- boot(data=train, statistic=RF_function, R=20, formula=Class ~.)
  e = mean_list(RF_reps$t)
  n_trees <- append(n_trees, tree)
  rf_error <- append(rf_error, e)
}
print(min(unlist(rf_error)))
rf_score_df = data.frame(unlist(n_trees),unlist(rf_error))
names(rf_score_df) = c("n_trees","Error")
ggplot(rf_score_df, aes(x=n_trees, y=Error)) +
  geom_line(color="#69b3a2", linewidth=1, alpha=0.9, linetype=1) +
```

```
  scale_x_continuous(breaks= trees) + ggtitle("No. of trees vs error")

# SVM
# Radial SVM with k-fold
data_list <- cv_fold(filtered_var_df, 7)
avg_error <- list()
for (i in data_list){
  cv_test <- as.data.frame(i)
  cv_train <- setdiff(filtered_var_df, cv_test)
  svm_classifier = svm(formula = Class ~ .,
                       data = cv_train,
                       type = 'C-classification',
                       kernel = 'radial',
                       cost = 5)
  svm_prediction <- predict(svm_classifier, cv_test)
  print(svm_prediction)
  error <- misclass(svm_prediction, cv_test$Class)
  avg_error <- append(avg_error, error)
}
mean_list(avg_error)

# SVM with bootstrapping
SVM_function <- function(formula, data, indices){
  d <- data[indices, ]
  svm_classifier = svm(formula = formula,
                       data = d,
                       type = 'C-classification',
                       kernel = 'radial',
                       cost = 5)
  svm_prediction <- predict(svm_classifier, test)
  return(misclass(svm_prediction, test$Class))
}
SVM_reps <- boot(data=train, statistic=SVM_function, R=20, formula=Class ~.)
mean_list(SVM_reps$t)

# GLM Boost
# GLM Boost with k-fold
avg_error <- list()
for (i in data_list){
  cv_test <- as.data.frame(i)
  cv_train <- setdiff(filtered_var_df, cv_test)
  GLMB_classifier = glmboost(Class ~ ., data = cv_train)
  GLMB_prediction <- predict(GLMB_classifier, newdata = cv_test)
  GLMB_prediction <- case_when(
     GLMB_prediction < 0.5 ~ 0,
```

```
    TRUE ~ 1)
  error <- misclass(GLMB_prediction, cv_test$Class)
  avg_error <- append(avg_error, error)
}
mean_list(avg_error)

# GLM Boost with bootstrapping
GLMB_function <- function(formula, data, indices){
  d <- data[indices, ]
  GLMB_classifier = glmboost(formula, data = d)
  GLMB_prediction <- predict(GLMB_classifier, newdata = test)
  return(misclass(GLMB_prediction, test$Class))
}
GLMB_reps <- boot(data=train, statistic=GLMB_function, R=20, formula=Class ~.)
mean_list(GLMB_reps$t)

# XGBoost
# XGBoost with k-fold
avg_error <- list()
for (i in data_list){
  cv_test <- as.data.frame(i)
  cv_train <- setdiff(filtered_var_df, cv_test)
  r <- length(cv_train)
  XGB_classifier = xgboost(label = cv_train$Class, data = data.matrix(cv_train[,1:r-1]), max_depth = 3,
nround=1, set.seed(2315880))
  XGB_prediction <- predict(XGB_classifier, newdata = data.matrix(cv_test[,1:r-1]))
  XGB_prediction <- case_when(
    XGB_prediction < 0.5 ~ 0,
    TRUE ~ 1)
  error <- misclass(XGB_prediction, cv_test$Class)
  avg_error <- append(avg_error, error)
}
mean_list(avg_error)

# XGBoost with bootstrapping
XGB_function <- function(formula, data, indices){
  d <- data[indices, ]
  r <- length(d)
  XGB_classifier = xgboost(label = formula, data = data.matrix(d[,1:r-1]), max_depth = 2, nround=3,
set.seed(2315880))
  XGB_prediction <- predict(XGB_classifier, newdata = data.matrix(test[,1:r-1]))
  XGB_prediction <- case_when(
    XGB_prediction < 0.5 ~ 0,
    TRUE ~ 1)
  return(misclass(XGB_prediction, test$Class))
```

```
}
XGB_reps <- boot(data=train, statistic=XGB_function, R=20, formula=train$Class)
mean_list(XGB_reps$t)
```

 # Question 4:

```
# Taking 20 principal components from second question
#Adding class column to filtered data
pca_data <- as.data.frame(pca.fit$score[,1:20])  # for imbalanced data
#pca_data <- as.data.frame(pca.fit$score[,1:25])  # For oversampled data
pca_data$Class <- new_df$Class
# shuffle the dataframe by rows
set.seed(2315880)
pca_data= pca_data[sample(1:nrow(pca_data)), ]
sample = sample.split(pca_data$Class, SplitRatio = 0.8)
train = subset(pca_data, sample == TRUE)
test  = subset(pca_data, sample == FALSE)
print(dim(train))
print(dim(test))

# KNN with k-fold
data_list <- cv_fold(pca_data, 7)
k = seq(1, 20, by=1)
k_value <- list()
cv_error <- list()
for (val in k)
{
  avg_error <- list()
  for (i in data_list){
    cv_test <- as.data.frame(i)
    cv_train = setdiff(pca_data, cv_test)
    classifier_knn <- knn(train = cv_train,
                          test = cv_test,
                          cl = cv_train$Class,
                          k = val,
                          set.seed(2315880))
    error <- misclass(classifier_knn, cv_test$Class)
    avg_error <- append(avg_error, error)
  }
  k_value <- append(k_value, val)
  e <- mean_list(avg_error)
```

```
  cv_error <- append(cv_error, e)
}
print(min(unlist(cv_error)))
knn_score_df = data.frame(unlist(k_value),unlist(cv_error))
names(knn_score_df) = c("k","cv_error")
ggplot(knn_score_df, aes(x=k, y=cv_error)) +
  geom_line( color="#69b3a2", linewidth=1, alpha=0.9, linetype=1) +
  scale_x_continuous(breaks= k) +
  ggtitle("Effect of no. of neighbours on error")

# KNN with bootstrapping
knn_function <- function(data, indices){
  d <- data[indices, ]
  classifier_knn <- knn(train = d,
                        test = test,
                        cl = d$Class,
                        k = val,
                       set.seed(2315880))
  return(misclass(classifier_knn, test$Class))
}
k = seq(1,20, by=1)
k_value <- list()
cv_error <- list()
for (val in k)
{
  KNN_reps <- boot(data=train, statistic=knn_function, R=20)
  ee <- mean_list(KNN_reps$t)
  k_value <- append(k_value, val)
  cv_error <- append(cv_error, ee)
  }
print(min(unlist(cv_error)))
knn_score_df = data.frame(unlist(k_value),unlist(cv_error))
names(knn_score_df) = c("k","cv_error")
ggplot(knn_score_df, aes(x=k, y=cv_error)) +
  geom_line( color="#69b3a2", linewidth=1, alpha=0.9, linetype=1) +
  scale_x_continuous(breaks= k) +
  ggtitle("Effect of no. of neighbours on error")

# Using hierarchical clusters
# Adding Clusters
h_clusters <- cutree(h_cluster_centroid, k = 2)
cluster_data <- filtered_df # for t-test reduced data
#cluster_data <- lasso_reduced # For lasso reduced data
cluster_data$clusters <- h_clusters
# shuffle the dataframe by rows
```

```
set.seed(2315880)
cluster_data= cluster_data[sample(1:nrow(cluster_data)), ]
sample = sample.split(cluster_data$Class, SplitRatio = 0.8)
train = subset(cluster_data, sample == TRUE)
test  = subset(cluster_data, sample == FALSE)
print(dim(train))
print(dim(test))


# KNN with k-fold
data_list <- cv_fold(cluster_data, 7)
k = seq(1, 20, by=1)
k_value <- list()
cv_error <- list()
for (val in k)
{
  avg_error <- list()
  for (i in data_list){
    cv_test <- as.data.frame(i)
    cv_train = setdiff(cluster_data, cv_test)
    classifier_knn <- knn(train = cv_train,
                          test = cv_test,
                          cl = cv_train$Class,
                          k = val,
                          set.seed(2315880))
    error <- misclass(classifier_knn, cv_test$Class)
    avg_error <- append(avg_error, error)
  }
  k_value <- append(k_value, val)
  e <- mean_list(avg_error)
  cv_error <- append(cv_error, e)
}
print(min(unlist(cv_error)))
knn_score_df = data.frame(unlist(k_value),unlist(cv_error))
names(knn_score_df) = c("k","cv_error")
ggplot(knn_score_df, aes(x=k, y=cv_error)) +
  geom_line( color="#69b3a2", linewidth=1, alpha=0.9, linetype=1) +
  scale_x_continuous(breaks= k) +
  ggtitle("Effect of no. of neighbours on error")

# KNN with bootstrapping
knn_function <- function(data, indices){
  d <- data[indices, ]
  classifier_knn <- knn(train = d,
                        test = test,
```

```
                      cl = d$Class,
                      k = val,
                     set.seed(2315880))
  return(misclass(classifier_knn, test$Class))
}
k = seq(1,20, by=1)
k_value <- list()
cv_error <- list()
for (val in k)
{
  KNN_reps <- boot(data=train, statistic=knn_function, R=20)
  ee <- mean_list(KNN_reps$t)
  k_value <- append(k_value, val)
  cv_error <- append(cv_error, ee)
  }
print(min(unlist(cv_error)))
knn_score_df = data.frame(unlist(k_value),unlist(cv_error))
names(knn_score_df) = c("k","cv_error")
ggplot(knn_score_df, aes(x=k, y=cv_error)) +
  geom_line( color="#69b3a2", linewidth=1, alpha=0.9, linetype=1) +
  scale_x_continuous(breaks= k) +
  ggtitle("Effect of no. of neighbours on error")


# Using k-means clusters
# Adding Clusters
k_clusters <- model$cluster
cluster_data <- filtered_df # for t-test reduced data
#cluster_data <- lasso_reduced # for lasso reduced data
cluster_data$clusters <- k_clusters
# shuffle the dataframe by rows
set.seed(2315880)
cluster_data= cluster_data[sample(1:nrow(cluster_data)), ]
sample = sample.split(cluster_data$Class, SplitRatio = 0.8)
train = subset(cluster_data, sample == TRUE)
test  = subset(cluster_data, sample == FALSE)
print(dim(train))
print(dim(test))

# KNN with k-fold
data_list <- cv_fold(cluster_data, 7)
k = seq(1, 20, by=1)
k_value <- list()
cv_error <- list()
for (val in k)
```

```
{
  avg_error <- list()
  for (i in data_list){
    cv_test <- as.data.frame(i)
    cv_train = setdiff(cluster_data, cv_test)
    classifier_knn <- knn(train = cv_train,
                          test = cv_test,
                          cl = cv_train$Class,
                          k = val,
                          set.seed(2315880))
    error <- misclass(classifier_knn, cv_test$Class)
    avg_error <- append(avg_error, error)
  }
  k_value <- append(k_value, val)
  e <- mean_list(avg_error)
  cv_error <- append(cv_error, e)
}
print(min(unlist(cv_error)))
knn_score_df = data.frame(unlist(k_value),unlist(cv_error))
names(knn_score_df) = c("k","cv_error")
ggplot(knn_score_df, aes(x=k, y=cv_error)) +
  geom_line( color="#69b3a2", linewidth=1, alpha=0.9, linetype=1) +
  scale_x_continuous(breaks= k) +
  ggtitle("Effect of no. of neighbours on error")

# KNN with bootstrapping
knn_function <- function(data, indices){
  d <- data[indices, ]
  classifier_knn <- knn(train = d,
                        test = test,
                        cl = d$Class,
                        k = val,
                       set.seed(2315880))
  return(misclass(classifier_knn, test$Class))
}
k = seq(1,20, by=1)
k_value <- list()
cv_error <- list()
for (val in k)
{
  KNN_reps <- boot(data=train, statistic=knn_function, R=20)
  ee <- mean_list(KNN_reps$t)
  k_value <- append(k_value, val)
  cv_error <- append(cv_error, ee)
  }
```

```
print(min(unlist(cv_error)))
knn_score_df = data.frame(unlist(k_value),unlist(cv_error))
names(knn_score_df) = c("k","cv_error")
ggplot(knn_score_df, aes(x=k, y=cv_error)) +
  geom_line( color="#69b3a2", linewidth=1, alpha=0.9, linetype=1) +
  scale_x_continuous(breaks= k) +
  ggtitle("Effect of no. of neighbours on error")
```