University of Essex

MA321-7 Team Project assignment

Analysis of gene expressions on invasive vs non-invasive cancer

Team D Group 2

Date :15.03.2024

Abstract:

The main aim of this project is to analyze gene expression on invasive vs non-invasive using machine learning algorithms in R language. The report summarizes the initial data preprocessing and analysis of the different models of both supervised and unsupervised models. And find the best machine learning model to analyze the gene expression of invasive and non-invasive cancer. We have

implemented a new model to try to improvise our best-supervised machine learning model using an unsupervised machine learning model (k clustering)

Word count: 2399

Table of Contents

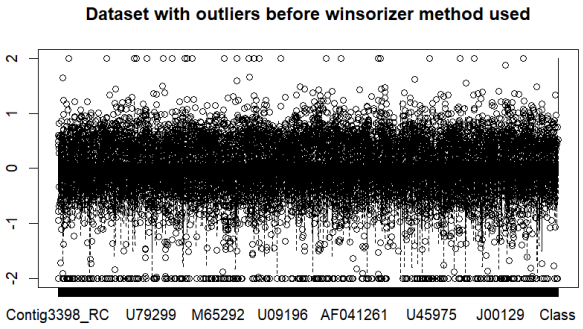
Introduction:

Our initial task is to analysis on the data set and identify the data type, distribution of data, and analysis of various variables. As the data set is huge, we have implemented PCA for unsupervised learning models and t-test for supervised learning models. A list of machine learning models of both supervised and unsupervised models was implemented. Using resampling techniques, the different machine learning models were compared and the best machine learning was identified. On our research question, we tried to improvise the clustering using our best machine learning model.

Analysis

Descriptive and graphical analysis

We are analyzing numerical data representing gene expressions in both invasive and non-invasive cancer genes. Initially, the dataset consists of 4949 variables indicating different genes among 78 patients, identified using the dim() function. We utilized the largest registered number within our team (2312181) as a seed to generate uniform random numbers from the provided CSV file. Subsequently, we created a subset consisting of 4948 randomly selected and ranked numbers, from which we extracted the first 2000 as our subset, referred to as 'team_gene_subset'. Upon inspection, we discovered 57 missing values within the dataset, which we addressed by replacing them with the median of non-missing values in each respective column. To manage outliers within the dataset, we employed the Winsorizer method. Initially, around 1602 variables exhibited outliers, which we successfully reduced to 245, thereby improving the dataset by minimizing the presence of outliers.



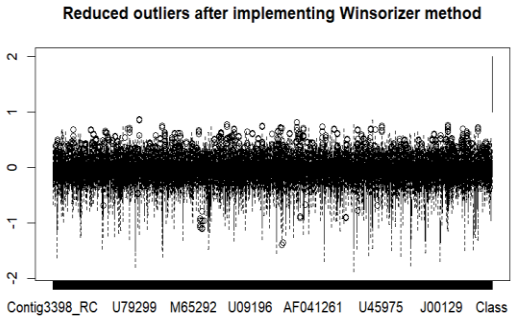


Fig 1: Dataset with outliers before Winsorizer method used

Fig 2: Dataset with reduced outliers after implementing Winsorizer method

Dimensional Reduction

Dimension reduction is a distinct technique used in data preprocessing to reduce the dimensions of the dataset. This helps to reduce the number of features while retaining the most important information. We have used Principal component analysis to reduce dimensions for unsupervised learning models and a two-sample t-test to reduce dimensions for supervised learning models.

PCA

PCA is used as dimensional reduction here as it retains the maximum originality of the data and maximum variance in the first few components, reducing the noise by focusing on the direction of maximum variance. PCA reduces dimensions by selecting crucial PCs, retaining much original data. The high cumulative variance explained suggests effective dimensionality reduction. Initial principal components PC1, PC2, and PC3 show higher variability and explain more variance than subsequent ones.

t-Test

A sample t-test is performed comparing the gene expression levels between two classes or groups represented by the "Class" variable. Genes with significant differences in expression levels between invasive and non-invasive cancer (class) are retained, reducing the dataset's dimensionality.

Unsupervised learning model

Machine learning models like PCA, k-means clustering, hierarchical clustering, and additionally t-SNE (t-distributed stochastic neighbor embedding) have been trained using the dimensional reduced dataset.

Below are the inferences of each model:

PCA:

The significance of principal components (PCs) is determined by their ranking, where PC1, having the highest standard deviation and explaining the most variance, holds the most significance, followed by PC2, PC3, and so forth. PC1's standard deviation of 1.41421 exceeds that of subsequent components, signifying its capture of the greatest variability in the dataset. Each PC's proportion of variance reveals its contribution to the overall dataset variance, with PC1 explaining 2.564% followed by PC2 with 1.282%, and so on. The cumulative proportion of variance demonstrates the combined impact of each PC, aiding in understanding information preservation as more PCs are considered. PC1 through PC22 collectively explain a significant portion of the dataset's variance, indicating their capture of fundamental patterns. These components are crucial for dimensionality reduction and feature selection, offering substantial information retention while reducing dimensionality. The PCA

model supports applications like dimensionality reduction, visualization, and feature selection, aiding in identifying significant genes or features. Researchers can determine the optimal number of retained principal components based on desired information retention levels. PC78, with a near-zero standard deviation and negligible variance, holds minimal relevance and can be disregarded in further analyses. In conclusion, PCA analysis provides valuable insights into dataset structure, guiding future investigations related to invasive and non-invasive cancer genes.

k-mean clustering

We see distinct two clusters formed. The optimal k value is 2, identified using the Silhouette method. This method provides the optimal score of the k value, and considers both cohesion and separation, providing a comprehensive measure of cluster quality rather than other methods focus on cluster variances. We see two distinct clusters formed with k=2.

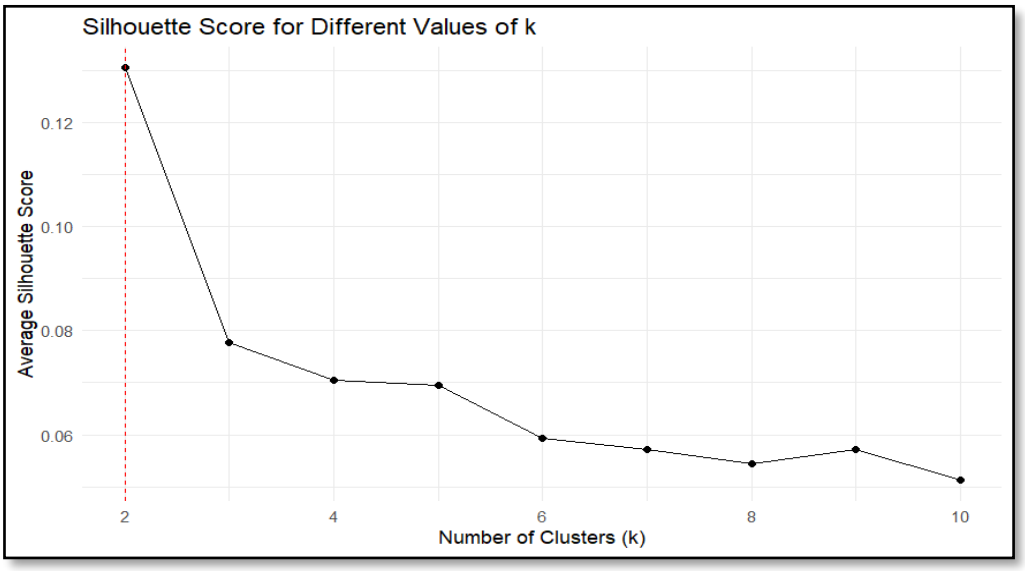


Fig 3 Using Silhouette score optimal k value is identified as 2

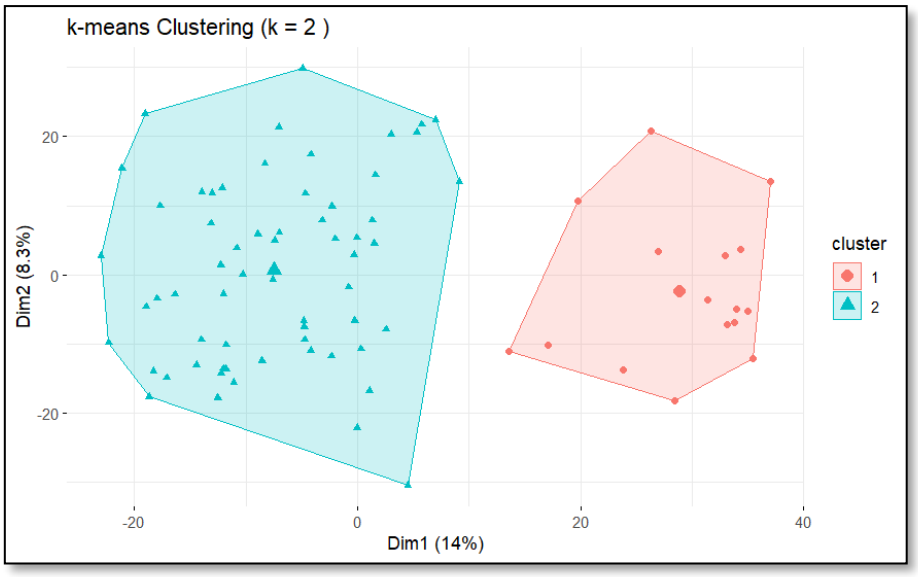


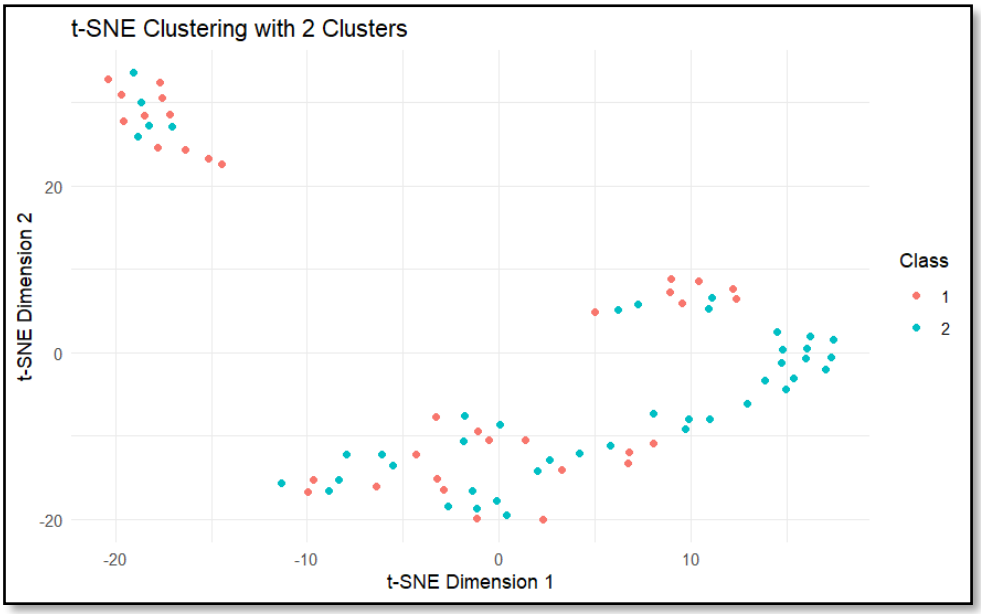
Fig 4: Distinct two clusters are formed with k- cluster model

Hierarchical clustering

The hierarchical clustering employed the complete linkage method and Euclidean distance to analyze 2000 genes representing patients. This method forms clusters based on maximum inter-cluster distance, yielding compact clusters. It organizes data into a hierarchical structure, grouping similar patients. In summary, hierarchical clustering offers a structured method for uncovering the dataset's inherent organization. It enables the recognition of specific patient clusters sharing similar gene expression patterns linked to both invasive and noninvasive cancer genes.

t-SNE(additional model)

We have an additional unsupervised machine learning model t-distributed stochastic neighbor embedding to visualize high dimensional data given each data point with a location. We have set the parameters PCA as a matrix excluding the last column. The t-SNE analysis utilized OpenMP with 1 thread, 2 dimensions, perplexity 10, and theta 0.5. It computed similarities, built a tree, and learned to embed. Error decreased to 0.231252 over 1000 iterations. Fitting took 0.13 seconds. The below visualization shows the dimensions of two clusters using t-SNE model



Supervised learning model

Supervised models are trained and predicted using the reduced dimensional dataset from a two-sample t-test. We have set the seed to ensure a random process, with the class label "Y" and predictor label "X" assigned to the matrix labelled "signi_gene_matrix" which contains the features related to the classification task. The data is split into training and testing as an 80:20 ratio ensuring model performance is evaluated on unseen data. 3-fold cross-validation is performed dividing the data into 3 subsets for training and validation. We have set accuracy as the default e-evaluation metric. The models consider 64 samples and 343 predictors to predict the classes. Sample size ranges from 42,43,43 respectively.

Logistic Regression

The model achieved an accuracy rate of approximately 51.52%. Regarding agreement between observed and predicted classes, the Kappa statistic measured at 0.004602. The F1 score, which balances precision and recall, stood at 0.416122. Sensitivity, indicating the proportion of actual positives correctly identified, was 39.26%, while specificity, representing the proportion of actual negatives correctly identified, reached 61.11%. The precision for positive cases was 44.44%, with a negative predictive value of 56.23%. The detection rate, or the proportion of actual positives correctly classified, was 17.17%. Considering class imbalance, the balanced accuracy was calculated at 50.19%. Overall, the logistic regression model shows potential for improvement, as its accuracy slightly surpasses random guessing.

LDA(Linear Discriminant Analysis)

The accuracy of the LDA model without implementing any preprocessing technique is 78.07%. Kappa is 0.5642 agreement between the predicted and actual class which is a moderate measure of agreement that accounts for the agreement occurring by chance alone. The F1 score is 0.7652174 suggests balanced precision and recall, indicating effective classification. Specificity (0.778) identifies noninvasive cancer genes accurately, aiding true negative identification. A positive Predictive Value (0.7904) shows the Likelihood of correctly identifying invasive cancer genes among positive predictions. Balanced Accuracy (0.7815) offers balanced classification accuracy considering class imbalances. The LDA model shows potential in distinguishing invasive and noninvasive cancer genes, with decent accuracy and balanced metrics.

QDA (Quadratic Discriminant Analysis)

Our QDA model was unfit for our random dataset as the model was unable to run a small group. The error pops up when the covariance matrices for each class are calculated.

k-NN(k- Nearest Neighbors)

The model with k = 5 was chosen as optimal due to its highest accuracy. The achieved accuracy value is 71.86%, which indicates the proportion of correctly classified cases. Kappa indicates moderate agreement between predicted and actual classification. The F1 score shows a good balance between precision and recall. The sensitivity (recall) is approximately 72.22%, indicating the ability to correctly detect invasive cancer genes. The accuracy (Pos_Pred_Value) is approximately 71.46%, which reflects the proportion of true invasive cancer gene predictions among all positive predictions. Neg_Pred_Value is 78.84. % indicating the proportion of true non-invasive cancer predictions out of all negative predictions. The average sensitivity and specificity reaches about 73.61%.

Random Forest

The random forest model achieved an accuracy of 75.04 with mtry=2, effectively classifying invasive and noninvasive cancer genes. However, for mtry values of 172 and 343, accuracy dropped to 0.64. Despite demonstrating reasonable sensitivity and specificity, additional tuning and validation can support better performance.

SVM (Support Vector Machine)

The SVM model achieved an accuracy of 73.3%, indicating its proficiency in classifying invasive and non-invasive cancer genes. With a kappa value of 0.522, the model exhibits moderate agreement beyond chance. An F1 score of 0.736 suggests a balance between precision and recall, enhancing its overall performance. The model demonstrates a sensitivity of 0.778 and a specificity of 0.750, indicating its ability to accurately identify both types of cancer genes. Positive predictive value (PPV) and negative predictive value (NPV) stand at 0.700 and 0.826, respectively, reflecting the model's predictive capability for positive and negative cases. A detection rate of 0.342 indicates the model's effectiveness in identifying positive cases relative to the total instances. With a balanced accuracy of 0.764, the model provides a fair estimate considering class distribution imbalance. The tuning parameter "C" remained constant at 1, suggesting consistent model performance across various regularization strengths.

GBM- (additional model)

The GBM model underwent evaluation with various tuning parameters such as interaction depth and the number of trees. Performance metrics including accuracy, Kappa statistic, F1 score, sensitivity, specificity, positive predictive value, and negative predictive value were analyzed across different parameter combinations. The optimal model, chosen based on accuracy, maintained a shrinkage parameter at 0.1 and a minimum of 10 observations in each terminal node. Accuracy ranged from approximately 69.05% to 83.33%, with Kappa statistics varying from 34.85% to 66.17%, and F1 scores from 58.27% to 80.99%. The optimal configuration featured an interaction depth of 3, 100 trees, shrinkage of 0.1, and a minimum of 10 observations in each terminal node. Overall, the GBM model shows promise in predicting invasive and noninvasive cancer genes.

Resampling technique – Cross validation

k-fold cross-validation techniques are used as a resampling technique in this dataset. It trains the unseen data by shuffling the data randomly and setting k groups. Each group acts as test data while the remaining groups act as training data. Later fit a model on the training set and evaluate it on the test set. Generally, we started k=10 to train the data however we could see the accuracy of all supervised models tend to increase as the k value decreased. We could k=3 gave the maximum accuracy of the models. This finding suggests that with a lower number of folds in the cross-validation process, the

model tends to generalize better to unseen data.In other words, by reducing the number of folds, the model captures more robust patterns in the data, resulting in higher accuracy.

Best model and implementation of k cluster to improve best model

The best model identified as GBM (Gradient Boosting Machine) which shows 83.33% while using the 3-fold cross-validation technique. GBM, short for Gradient Boosting Machine, is a potent algorithm in machine learning recognized for its exceptional predictive precision and resilience. Within the provided table, GBM showcases impressive performance across diverse evaluation criteria. It attains an average accuracy of around 83.33%, positioning it as one of the leading models in the comparison. Furthermore, GBM demonstrates elevated values for several other metrics including Balanced Accuracy, F1 Score, Kappa, Sensitivity, and Specificity, underscoring its proficiency in precisely distinguishing between invasive and non-invasive genes. These findings underscore GBM's potential as a promising option for predictive modeling endeavors in gene expression analysis.

Metric	GBM	LDA	Random Forest	SVM	KNN	Logistics Regression
Accuracy	83.33	78.07	75.04	73.3	71.86	51.52

Table 1: Summary of accuracy of supervised learning models in descending order

As per our research question, we have implemented our k cluster model to check if our GBM model is improvising. The model achieved an accuracy rate of 85.71%, suggesting that it accurately classified around 85.71% of instances into their respective categories (invasive or non-invasive cancer). With a Kappa statistic of 0.6957, there is substantial agreement between the model's predictions and the actual classes, surpassing what would be expected by chance alone. Sensitivity, at 66.67%, indicates the model's ability to correctly identify 66.67% of true positive cases (invasive cancer) among all actual positive cases. Specificity is at 100%, meaning the model accurately identifies all true negative cases (non-invasive cancer) among all actual negative cases. A positive predictive value of 1 implies that the model correctly predicts instances as positive (invasive cancer) 100% of the time. Conversely, a negative predictive value of 0.8 indicates that the model correctly predicts instances as negative (non-invasive cancer) 80% of the time. The dataset exhibits a prevalence of invasive cancer at approximately 42.86%. The model successfully detects around 28.57% of all invasive cancer cases.

Metric	Log Regr	LDA	GBM	KNN	Random Forest	SVM
Balanced Accuracy						
Min.	0.417	0.694	0.806	0.694	0.722	0.667
1st Qu.	0.475	0.756	0.819	0.710	0.724	0.681
Median	0.533	0.817	0.833	0.725	0.725	0.694
Mean	0.502	0.781	0.833	0.723	0.746	0.726
3rd Qu.	0.544	0.825	0.847	0.738	0.758	0.756
Max.	0.556	0.833	0.861	0.750	0.792	0.817
Detection Rate						
Min.	0.143	0.286	0.333	0.286	0.286	0.238
1st Qu.	0.162	0.325	0.345	0.302	0.302	0.262
Median	0.182	0.364	0.357	0.318	0.318	0.286
Mean	0.172	0.343	0.357	0.328	0.312	0.296
3rd Qu.	0.186	0.372	0.369	0.350	0.326	0.325
Max.	0.190	0.381	0.381	0.381	0.333	0.364
F1						
Min.	0.333	0.696	0.778	0.696	0.700	0.625
1st Qu.	0.389	0.748	0.794	0.698	0.700	0.628
Median	0.444	0.800	0.810	0.700	0.700	0.632
Mean	0.416	0.765	0.810	0.701	0.717	0.686
3rd Qu.	0.458	0.800	0.826	0.703	0.725	0.716
Max.	0.471	0.800	0.842	0.706	0.750	0.800
Kappa						
Min.	-0.167	0.364	0.611	0.364	0.432	0.329
1st Qu.	-0.049	0.498	0.636	0.407	0.441	0.364
Median	0.068	0.633	0.662	0.450	0.450	0.400
Mean	0.005	0.564	0.662	0.440	0.494	0.454
3rd Qu.	0.090	0.664	0.687	0.479	0.525	0.517
Max.	0.113	0.696	0.712	0.507	0.600	0.633
Neg_Pred_Value						
Min.	0.500	0.800	0.833	0.750	0.750	0.714
1st Qu.	0.536	0.817	0.852	0.760	0.768	0.721
Median	0.571	0.833	0.871	0.769	0.786	0.727
Mean	0.562	0.830	0.871	0.792	0.779	0.758
3rd Qu.	0.593	0.845	0.890	0.813	0.793	0.780
Max.	0.615	0.857	0.909	0.857	0.800	0.833
Pos_Pred_Value						
Min.	0.333	0.571	0.778	0.571	0.636	0.600
1st Qu.	0.417	0.686	0.783	0.636	0.668	0.657
Median	0.500	0.800	0.789	0.700	0.700	0.714
Mean	0.444	0.790	0.789	0.674	0.731	0.705
3rd Qu.	0.500	0.900	0.794	0.725	0.779	0.757

Max.	0.500	1.000	0.800	0.750	0.857	0.800
Precision						
Min.	0.333	0.571	0.778	0.571	0.636	0.600
1st Qu.	0.417	0.686	0.783	0.636	0.668	0.657
Median	0.500	0.800	0.789	0.700	0.700	0.714
Mean	0.444	0.790	0.789	0.674	0.731	0.705
3rd Qu.	0.500	0.900	0.794	0.725	0.779	0.757
Max.	0.500	1.000	0.800	0.750	0.857	0.800
Recall						
Min.	0.333	0.667	0.778	0.667	0.667	0.556
1st Qu.	0.367	0.733	0.806	0.683	0.683	0.611
Median	0.400	0.800	0.833	0.700	0.700	0.667
Mean	0.393	0.785	0.833	0.752	0.715	0.674
3rd Qu.	0.422	0.844	0.861	0.794	0.739	0.733
Max.	0.444	0.889	0.889	0.889	0.778	0.800
Sensitivity						
Min.	0.333	0.667	0.778	0.667	0.667	0.556
1st Qu.	0.367	0.733	0.806	0.683	0.683	0.611
Median	0.400	0.800	0.833	0.700	0.700	0.667
Mean	0.393	0.785	0.833	0.752	0.715	0.674
3rd Qu.	0.422	0.844	0.861	0.794	0.739	0.733
Max.	0.444	0.889	0.889	0.889	0.778	0.800
Specificity						
Min.	0.500	0.500	0.833	0.500	0.667	0.667
1st Qu.	0.583	0.667	0.833	0.625	0.708	0.750
Median	0.667	0.833	0.833	0.750	0.750	0.833
Mean	0.611	0.778	0.833	0.694	0.778	0.778
3rd Qu.	0.667	0.917	0.833	0.792	0.833	0.833
Max.	0.667	1.000	0.833	0.833	0.917	0.833

Table 2: Summary of key metrics of 6 supervised learning model

Conclusion

Invasive and non-invasive cancer genes have different gene expression patterns, which are crucial for understanding cancer progression. This report analyses the predictions of the Gradient Boosting Machine (GBM) model to classify gene expression as invasive or non-invasive with accuracy of 85.71% which has improved to 2.38% from the initial GBM model.

In summary, the GBM model, when trained on gene expression data alongside supplementary cluster features, shows encouraging capabilities in discerning between invasive and non-invasive cancer types. The model showcases notable accuracy, significant concordance with real classes, and favourable sensitivity and specificity.

References:

- An Introduction to Statistical Learning with Applications in R- Gareth ,James Daniela Witten,Trevor Hastie ,Robert Tulshiram
- https://www.datacamp.com/tutorial/pca-analysis-r
- https://www.datacamp.com/tutorial/k-means-clustering-r
- https://www.datacamp.com/tutorial/machine-learning-in-r

Appendix

```
# Load necessary library

library(stats)

library(pls)

library(caret)

library(Rtsne)

library(tidymodels)

library(themis)

library(tidyverse)

library(ggplot2)

library(MASS)

library(gbm)

library(class)

library(randomForest)

library(e1071)

library(nnet)

library(dplyr)

library(dbscan)

library(factoextra)
```

```
library(cluster)

library(tidyr)

# Read the CSV file 'gene-expression-invasive-vs-noninvasive-cancer.csv' into a dataframe 'InitialData'

 InitialData <- read.csv(file="D:/R Lab/Group Project/Coursework Initial Task-20240304/gene-expression-invasive-vs-noninvasive-cancer.csv")

   # Set the seed for reproducibility

 set.seed(2312181)

   # Generate 4948 random numbers, rank them, and select the first 2000 as indices for the gene subset

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

 # Add an additional index (4949) to the selected gene subset

 team_gene_subset <- c(team_gene_subset, 4949)

   # Subset the 'InitialData' dataframe using the selected gene indices

 team_gene_subset <- InitialData[, team_gene_subset]

# Generate random gene expression data

 set.seed(2312181)  # for reproducibility

 gene_data <- matrix(rnorm(2000*50), ncol = 50)  # 2000 genes, 50 samples

   # Convert the matrix to a data frame

 gene_data_df <- as.data.frame(gene_data)

   # Create a box plot for the gene expression data

 boxplot(gene_data_df, main = "Gene Expression Data", xlab = "Samples", ylab = "ExpressionLevel")

library(dplyr)

sum(is.na(team_gene_subset))
```

```
# Define a function for median imputation
impute_median <- function(x) {
  median_value <- median(x, na.rm = TRUE)
  replace(x, is.na(x), median_value)
}
# Apply median imputation to replace missing values
team_gene_subset_imputed_median <- team_gene_subset %>%
  mutate(across(everything(), impute_median))
# Display the imputed data for median
print(team_gene_subset_imputed_median)
# Set the number of genes for plotting
num_genes <- 10  # Adjust the number of genes to plot as desired
# Select a subset of genes for plotting
genes_subset <- team_gene_subset_imputed_median[, 1:num_genes]
# Convert the data to long format for plotting
genes_subset_long <- stack(genes_subset)
# Plot the boxplot
boxplot(values ~ ind, data = genes_subset_long,
        main = "Gene Expression Boxplot (After Median Imputation)",xlab = "Genes", ylab =
"Expression Level")
#To show outliers before winsorizering
boxplot(team_gene_subset_imputed_median, main="Dataset with outliers before winsorizer method
used")
# to identify the no. of genes expressions in outliers
# Load necessary libraries
```

```
library(zoo)
library(dplyr)
# Sample data has been imputed with median imputation
# Assuming 'team_gene_subset_imputedmedian' is your dataset after imputation
# Function to detect outliers in a vector
detect_outliers <- function(x) {
  qnt <- quantile(x, probs=c(.25, .75), na.rm = TRUE)
  iqr <- IQR(x, na.rm = TRUE)
  outliers <- x < (qnt[1] - 1.5 * iqr) | x > (qnt[2] + 1.5 * iqr)
  return(outliers)
}
# Apply the function to each column of the dataset
outlier_columns <- lapply(team_gene_subset_imputedmedian, detect_outliers)
# Identify columns with outliers
cols_with_outliers <- sum(sapply(outlier_columns, any))
cols_with_outliersnam <- names(which(sapply(outlier_columns, length) > 0))
# Print the number of columns with outliers
print(cols_with_outliers)
print(cols_with_outliersnam)
# to reduce outliers
library(DescTools)
# Sample data has been imputed with median imputation
# Assuming 'team_gene_subset_imputedmedian' is your dataset after imputation
# Function to perform winsorization on a vector
```

```
winsorize_column <- function(x, probs = c(0.05, 0.95), na.rm = FALSE) {
  Winsorize(x, probs = probs, na.rm = na.rm, type = 7)
}
# Apply winsorization to each column of the dataset
team_gene_subset_winsorized <- lapply(team_gene_subset_imputed_median, winsorize_column)
# Convert the list back to a data frame
team_gene_subset_winsorized <- as.data.frame(team_gene_subset_winsorized)
# Print the dataset after winsorization
print(team_gene_subset_winsorized)
boxplot(team_gene_subset_winsorized,main = "Reduced outliers after implementing Winsorizer method")
# to check how much outliers removed by winsorizer
# Count outliers removed from each column
outliers_removed <- sapply(1:length(team_gene_subset_imputed_median), function(i) {
  sum(team_gene_subset_imputed_median[[i]] != team_gene_subset_winsorized[[i]])
})
# Print the number of outliers removed for each column
print(outliers_removed)
# Convert the list back to a data frame
team_gene_subset_winsorized <- as.data.frame(team_gene_subset_winsorized)
# Calculate the count of outliers removed
outliers_before <- sum(team_gene_subset_imputed_median)  # Count outliers in the original dataset
outliers_before
outliers_after <- sum(team_gene_subset_winsorized)  # Count outliers in the winsorized dataset
outliers_after
```

```
outliers_removed <- outliers_before - outliers_after  # Calculate the difference
# Print the count of outliers removed
print(paste("Outliers removed by Winsorization:", outliers_removed))
# to identify the no. of genes expressions in outliers
# Load necessary libraries
library(zoo)
library(dplyr)
# Sample data has been imputed with median imputation
# Assuming 'team_gene_subset_imputedmedian' is your dataset after imputation
# Function to detect outliers in a vector
detect_outliersaf <- function(x) {
  qnt <- quantile(x, probs=c(.25, .75), na.rm = TRUE)
  iqr <- IQR(x, na.rm = TRUE)
  outliers <- x < (qnt[1] - 1.5 * iqr) | x > (qnt[2] + 1.5 * iqr)
  return(outliers)
}
# Apply the function to each column of the dataset
outlier_columnsaf <- lapply(team_gene_subset_winsorized, detect_outliersaf)
# Identify columns with outliers
cols_with_outliersaf <- sum(sapply(outlier_columnsaf, any))
cols_with_outliersnamaf <- names(which(sapply(outlier_columnsaf, length) > 0))
# Print the number of columns with outliers
print(cols_with_outliersaf)
print(cols_with_outliersnam)
```

```
# t test - Dimensional reduction

# Extract column names of team_gene_subset except the first and last column

gene_columns <- colnames(team_gene_subset_winsorized)[-c(1,
ncol(team_gene_subset_winsorized))]

# Define a function for performing t-tests

per_t_test <- function(data, gene_column) {

  # Check if there is sufficient variability in the data

  if (length(unique(data[[gene_column]])) <= 1) {

    # If there is no variability, return NA

    return(NA)

  } else {

    # Perform a two-sample t-test comparing the gene expression levels between two classes/groups

    t_res <- t.test(data[[gene_column]] ~ data$Class)

    # Return the p-value obtained from the t-test

    return(t_res$p.value)

  }

}

# Perform t-test for each gene column in the dataset

p_values <- sapply(gene_columns, per_t_test, data = team_gene_subset_winsorized)

# Identify genes with p-values less than 0.05, indicating statistical significance

signi_gene <- gene_columns[p_values < 0.05]


# Combine the significant genes with the last column ('Class') to include the class labels

signi_gene_with_lbl <- c(signi_gene, "Class")

# Extract the relevant columns from your original dataset
```

```
signi_gene_data <- team_gene_subset_winsorized[, c(signi_gene_with_lbl)]
# Convert class labels to a factor for classification
signi_gene_data$Class <- as.factor(signi_gene_data$Class)
# Create a matrix using the significant genes data
signi_gene_matrix <- as.matrix(signi_gene_data[, -ncol(signi_gene_data)])
```

```{r}
# Perform PCA
pca_result <- prcomp(team_gene_subset_winsorized, scale. = TRUE)
# Summary of PCA
summary(pca_result)
# Variance explained by each principal component
print(pca_result$sdev^2 / sum(pca_result$sdev^2))
# Extract principal components
PC <- as.data.frame(pca_result$x)
# Print the extracted principal components
print(PC)
# Performing PCA using the dimensional reduced PC components
# PCA using the PC components  excluding the first and last
pca_result1 <- prcomp(PC, center = TRUE, scale. = TRUE)
summary(pca_result1)
# Scree plot
plot(1:length(pca_result1$sdev), pca_result1$sdev^2, type = "b",
     xlab = "Principal Component", ylab = "Variance Explained",
```

```
    main = "Scree Plot for PCA")
# Extract PC scores
pc_scores <- as.data.frame(pca_result$x)
# Visualize PCA using factoextra
fviz_eig(pca_result1, addlables=TRUE)
# Performing PCA using the dimensional reduced PC components
# PCA using the PC components  excluding the first and last
pca_result1 <- prcomp(PC, center = TRUE, scale. = TRUE)
summary(pca_result1)
# Scree plot
plot(1:length(pca_result1$sdev), pca_result1$sdev^2, type = "b",
    xlab = "Principal Component", ylab = "Variance Explained",
    main = "Scree Plot for PCA")
# Extract PC scores
pc_scores <- as.data.frame(pca_result$x)
# Visualize PCA using factoextra
fviz_eig(pca_result1, addlables=TRUE)
```

```{r}
# K means clustering
# Define the silhouette_score function to determine the optimal k value
silhouette_score <- function(k, df) {
  km <- kmeans(df, centers = k, nstart = 25)
  ss <- silhouette(km$cluster, dist(df))
```

```
  mean(ss[, 3])
}
library(ggplot2)
# Define the range of k values
k_values <- 2:10
# Compute silhouette scores for each value of k
sil_scores <- sapply(k_values, silhouette_score, df = PC)
# Create a data frame for plotting
sil_data <- data.frame(k = k_values, silhouette_score = sil_scores)
# Plot silhouette scores
ggplot(sil_data, aes(x = k, y = silhouette_score)) +
  geom_line() +
  geom_point() +
  geom_vline(xintercept = 2, linetype = "dashed", color = "red") +
  labs(title = "Silhouette Score for Different Values of k",
       x = "Number of Clusters (k)",
       y = "Average Silhouette Score") +
  theme_minimal()
k = 2
# Perform k-means clustering with k = 2
kmeans_result <- kmeans(PC, centers = k, nstart = 25)
kmeans_result
# Visualize clustering results
fviz_cluster(kmeans_result, data = PC, geom = 'point',
```

```
        stand = FALSE, ellipse.type = "convex",
        ggtheme = theme_minimal(),
        main = paste('k-means Clustering (k =', k, ')'))
wss <- sum(kmeans_result$withinss)
print(paste("Within-cluster sum of squares (WSS):", wss))
}
#Hierarchial clustering
# Compute distance matrix using Euclidean distance
dist_mat <- dist(t(team_gene_subset_winsorized[, -ncol(team_gene_subset_winsorized)]))
# Perform hierarchical clustering
hc_result <- hclust(dist_mat, method = "complete")
hc_result
# Plot the dendrogram
plot(hc_result, main = "Hierarchical Clustering Dendrogram", sub = "", xlab = "",cex=0.1)
# Perform Principal Component Analysis (PCA) on the gene expression data, excluding the first and last columns
pca_result2 <- prcomp(team_gene_subset_winsorized[, -ncol(team_gene_subset_winsorized)], center = TRUE, scale. = TRUE)
# Extract the first two principal components
PC1 <- pca_result2$x[, 1]
PC2 <- pca_result2$x[, 2]

# Combine the first two principal components with the Class column
pca_data <- cbind(PC1, PC2, Class = team_gene_subset_winsorized$Class)
# If pca_data is a data frame and contains a column named 'Class'
```

```
# Proceed with creating tsne_data

# Load necessary library

library(Rtsne)

# Re-run t-SNE with 2 dimensions and a lower perplexity value

tsne_result_2_clusters <- Rtsne(as.matrix(pca_data[, -ncol(pca_data)]), dims = 2, perplexity = 10,
verbose = TRUE)

# Create a data frame with t-SNE results

tsne_data_2_clusters <- data.frame(PC1 = tsne_result_2_clusters$Y[, 1],

                                   PC2 = tsne_result_2_clusters$Y[, 2],

                                   Class = pca_data[, 3])

# Plot the t-SNE clusters with 2 clusters

ggplot(data = tsne_data_2_clusters, aes(x = PC1, y = PC2, color = factor(Class))) +

  geom_point() +

  labs(title = "t-SNE Clustering with 2 Clusters",

       x = "t-SNE Dimension 1",

       y = "t-SNE Dimension 2",

       color = "Class") +

  theme_minimal()

# Implemeting resampling technique - cross validation, set to train the models

# Set seed for reproducibility

set.seed(2312181)


# Convert class labels to a factor for classification

Y <- as.factor(signi_gene_data$Class)

X <- signi_gene_matrix
```

```
# Split the data into training and testing sets
trainIndex <- createDataPartition(Y, p = .8, list = FALSE)
X_train <- X[trainIndex, ]
Y_train <- Y[trainIndex]
X_test <- X[-trainIndex, ]
Y_test <- Y[-trainIndex]
# Cleaning factor levels of Y_train to ensure they are valid R variable names
levels(Y_train) <- make.names(levels(Y_train))
Y_train <- factor(Y_train)
# Set up cross-validation control with class probabilities
control <- trainControl(method = "cv",
                        number = 3,
                        summaryFunction = multiClassSummary,
                        savePredictions = TRUE)
# Define the metric for evaluation
metric <- "Accuracy"
# Logistic Regression
model_log <- train(Y_train ~ ., data = data.frame(X_train, Y_train),
                   method = "glm", family = "binomial",
                   trControl = control)
model_log
# LDA
model_lda <- train(Y_train ~ ., data = data.frame(X_train, Y_train), method = "lda", trControl =
control, metric = metric)
model_lda
```

```
#QDA

#model_qda <- train(Y_train ~ ., data = data.frame(X_train, Y_train), method = "gbm", trControl = control, metric = metric)

# GBM

model_gbm <- train(Y_train ~ ., data = data.frame(X_train, Y_train), method = "gbm", trControl = control, metric = metric, tuneLength = 7)

model_gbm


# KNN

model_knn <- train(Y_train ~ ., data = data.frame(X_train, Y_train), method = "knn", trControl = control, metric = metric, tuneLength = 7)

# Random Forest

model_rf <- train(Y_train ~ ., data = data.frame(X_train, Y_train), method = "rf", trControl = control, metric = metric, tuneLength = 3)

# SVM

model_svm <- train(Y_train ~ ., data = data.frame(X_train, Y_train), method = "svmLinear", trControl = control, metric = metric, tuneLength = 3)

# Create a list of model objects

model_list <- list(LogReg = model_log, LDA = model_lda, GBM = model_gbm, KNN = model_knn, RandomForest = model_rf, SVM = model_svm)


# Create the resamples object

results <- resamples(model_list)

# Analyze the results

summary(results)

# Implement kmeans clustering to check if the GBM model improvise.
```

```
set.seed(2312181)
k=2
kmeans_up <- kmeans(PC, centers = k, nstart = 25)
# Extracting cluster labels
label_cluster <- kmeans_up$cluster
# Add cluster labels as a new feature to the training data
X_train_with_clusters <- cbind(X_train, Cluster = label_cluster)
# Fit the GBM model with the additional cluster feature
model_gbm_with_clusters <- train(Y_train ~ .,
                                 data = data.frame(X_train_with_clusters, Y_train),
                                 method = "gbm",
                                 trControl = control,
                                 metric = metric,
                                 tuneLength = 5)
# Compare the performance of the models
print(summary(model_gbm))  # Summary of the original model without clusters
print(summary(model_gbm_with_clusters))  # Summary of the model with added cluster feature
# Predict using the model on the test data
predictions <- predict(model_gbm_with_clusters, newdata = data.frame(X_test, Cluster =
label_cluster[-trainIndex]))
levels(predictions) <- levels(Y_test)
# Confusion matrix
confusion_matrix <- confusionMatrix(predictions, Y_test)
print(confusion_matrix)
# Accuracy
```

```
accuracy <- confusion_matrix$overall["Accuracy"]

print(paste("Accuracy:", accuracy))
```