Understanding a Heart Disease dataset using supervised and unsupervised machine learning methods: A DAT121 Project

Abdirahman, Domantas, Ingebrigt, Maria, Muna

September 6, 2024

Abstract

In modern machine learning, diagnosing and forecasting heart disease using vast datasets and predictive patterns is central. We utilized supervised and unsupervised machine learning methods along with filtering and ensemble methods in feature selection on a multivariate Heart Disease dataset with 13 features and 1 response variable, containing 303 samples per feature. We compared the performance of our supervised learning methods and attempted to identify important features. SVM, kNN, Logistic Regression with Newton-Raphson's optimization and Random Forest were utilized as supervised learning methods, while a PCA was conducted as our sole unsupervised learning method, retaining 6 components, and setting up a dimensionality reduced dataset which our supervised methods were used on. This was crucial for improving the accuracy. The data was split into 80/20 training/test, and the supervised learning method with the best performance was identified, while results were displayed for each individual method and in a comparison table for all methods. We performed grid searches and cross-validation for parameters on different methods and applied model-dependent and model-independent filtering methods along with ensemble methods for feature selection, capturing a range of statistical relationships, but decided against discarding any features due to the selection of different features by different methods. We achieved the highest accuracy score with Logistic Regression, giving an accuracy score of 0.89 while also having a very low computational cost, with the Newton-Raphson optimization reaching convergence within 7 iterations. We obtained the lowest accuracy using Random Forest algorithm, with an accuracy score of 0.83. A GitHub repository for this project can be accessed from Project Git.

1 Introduction

Heart disease is well-known as one of the leading causes of mortality worldwide, with an estimation of 17.9 million lives lost per year [1]. Early detection and prevention are crucial in reducing the fatality rate associated with cardiovascular conditions. While traditional medical approaches have focused on risk factors like cholesterol levels and lifestyle habits, the rise of data-driven methods has introduced predictive analytics using machine learning. By analyzing complex datasets, these algorithms can uncover hidden patterns and relationships that may not be immediately apparent to healthcare professionals. The goal of this report is to develop and evaluate machine learning models for predicting the likelihood of heart attacks in patients. Using a dataset containing a variety of clinical and demographic factors, we explore the efficiency of several algorithms in identifying high-risk individuals. Through feature engineering, model optimization, and performance evaluation, we aim to demonstrate how data science can be a powerful tool in the fight against heart disease.

2 Theory and background

200 million people are living with coronary heart disease, with 49 million of them in the EU [2]. Cardiovascular disease cost the EU economy 282 billion Euros in 2021 [2], and statistically, it is rising in patients with age. A budding heart disease can often be kept in check with lifestyle changes and medication, but heart damage is irreversible and once extensive damage has been done, complex and risky surgery that is demanding on the body is often the only option, leaving patients with few alternatives to a painful and dooming life-or-death situation.

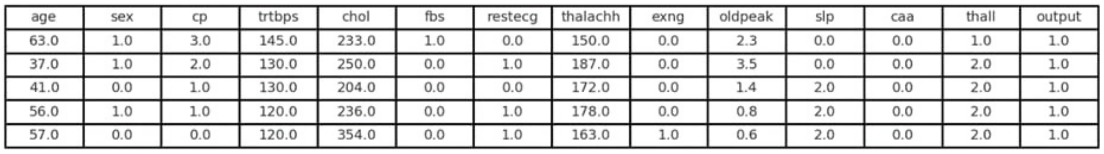
In cardiology and preventive healthcare, a main objective is to predict heart attacks before they occur. Early prediction makes it possible to significantly reduce the risk of severe outcomes, including death. Machine learning offers the ability to analyze complex datasets and generate precise predictions by training on historical data. By uncovering hidden patterns and providing new insights, machine learning models can help identify previously unknown risk factors, ultimately improving the detection of potential heart attack risks in patients.
When building a machine learning model, it is essential to evaluate its performance on data it hasnt seen during training, to simulate how the model would perform on real-world unseen data. Any further analysis, that is, in order to advance to clustering or classification, we are completely dependent on the data splitting being done in a way that maintains a balance between complexity, interpretability and reproducibility. [3].

By splitting the data into training and test sets, the model can be trained on one portion and evaluated on another. When a model learns not only the underlying patterns but also the noise in the training data, the model becomes overfitted and will perform well on the training data but not on unseen data. By keeping a separate test set, we can check if the model generalizes well to new data. When tuning hyperparameters, the performance on the test set can guide the selection of the best model.

Without a test set, there is a risk of tuning the model such that it performs well only on the training data but not on the new data. A test set gives an objective evaluation metric for the model, ensuring that performance metrics like accuracy, precision, and recall are not overly optimistic and reflect the models ability to generalize.

The Heart Attack Analysis and Prediction Dataset comes from Kaggle, a popular data science competition and data sharing site. A link to where the dataset was taken from can be accessed here: Kaggle. Rashik Rahman Pritorm, a data enthusiast who has added numerous datasets to the Kaggle community. The purpose of releasing the dataset was to assist with the analysis of heart attack risk variables and the development of machine learning models for prediction by researchers, learners and data scientists [4]. The data is collected from the Cleveland database and published by the University of California, Irvine (UCI), in 1988 [5].

2.1 Dataset Overview and structure



age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
63.0	1.0	3.0	145.0	233.0	1.0	0.0	150.0	0.0	2.3	0.0	0.0	1.0	1.0
37.0	1.0	2.0	130.0	250.0	0.0	1.0	187.0	0.0	3.5	0.0	0.0	2.0	1.0
41.0	0.0	1.0	130.0	204.0	0.0	0.0	172.0	0.0	1.4	2.0	0.0	2.0	1.0
56.0	1.0	1.0	120.0	236.0	0.0	1.0	178.0	0.0	0.8	2.0	0.0	2.0	1.0
57.0	0.0	0.0	120.0	354.0	0.0	1.0	163.0	1.0	0.6	2.0	0.0	2.0	1.0

Figure 1: **Table with the general structure of the heart-disease data containing a sample of observations, with features in the column-dimension, and observations in the row-dimension [5].**

The dataset consists of 303 observations per feature (rows) and 13 features (columns), each illustrating a crucial attribute connected to hearth health. Here is a breakdown of the features: The dataset has 303 observations, each of which representing a patient's heart health related to their medical profile. We also have 13 elements (columns), all of which are known to affect the risk of a heart attack. These 13 columns include clinical and demographic elements. This information offers a whole picture of several risk factors and health indicators connected. An explanation of these features and their meaning is given below:

Age: Cardiac diseases affected by age.
Sex: Explains heart diseases development by gender, Male(1) and Female(0).
Chest Pain Type (cp): Risk factors separate by chest pain type.
Resisting blood Pressure (trtbps): High resisting blood pleasure, which is a risk factor for heart diseases.
Cholesterol Level (chol): High cholesterol can cause heart diseases, making it an important feature.
Fasting blood sugar (fbs): High blood sugar is connected to diabetes, which raises heart diseases risk.
Resting ECG results (restec): Shows hearts electrical activity; anomalies may indicate cardiac issues.
Max Heart Rate (thalachh): Determines heart health by measuring cardiac response to exercise.
Exercies-Induced Angina (exng): Indicates whether the patient has chest pain during activity (1=yes, 0=no).
ST depression (oldpeak): an electrocardiogram depression indicating heart illness.
ST segment slope (slp): slope of the heart's electrical cycle seen on an ECG, measured after physical activity to assess the heart's health.
Number of major vessels (caa): The count of vital blood vessels, such as the arteries, that provide blood to the heart. Medically, 'caa' stands for coronary artery assessment, which checks these channels for irregularities.
Thalassemia (thall): a blood condition that can harm the heart.
target (output) is high risk of heart attack (1) or not high risk for heart attack (0).

The structure of dataset allows an in-depth review of every factors leading to heart disease. This dataset is a useful resource for developing models that predict since it includes medical indicators like heart rate, cholesterol and blood pleasure with demographic information as sex and age. Each feature has been carefully selected to provide insight into various aspects of heart health. This helps to make sure that the dataset covers every key area required for a full hearth disease analysis.

2.2 Logistic Regression algorithm

Logistic regression, a supervised learning method learns a general rule(Equation 1) mapping input to outputs by generalising from the training data to unseen situations. It was chosen as one of the classifiers due to its particular fit when dealing with binary classification problems. When using Logistic Regression it is crucial to assume linearity of the logit, meaning explanatory variables not having a linear relationship with the logit of the response variable, independent observations, no perfect multicollinearity [6] and it is important to have a large sample size. A key problem in logistic regression is that explanatory variables considered for the logistic regression model are highly correlated amongst themselves, so multicollinearity causes unstable estimates and inaccurate variances that affects confidence intervals. In this example, multicollinearity could contribute to overfitting, and features like "cholesterol levels" and "triglyceride levels" are highly correlated, so it could become difficult to determine which of these features are actually contributing to the prediction of heart disease. For this reason, principal component analysis was applied prior to the classification, representing one solution where the multicollinearity is not a potential issue, as principal components are linearly independent, taking collinearity out of the equation. This dataset has a large sample size of 303 samples per feature, with 13 features and 1 target variable, from here on referred to as y.

The formula used in logistic regression is:

$$\frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1}} \tag{1}$$

, where the target variable y indicates whether or not a patient has heart disease (binary outcome: 0 means no disease, 1 means disease).

The MLE(Maximum Likelihood Estimator), modelling a conditional expectation function, allows us to multiply the densities in the statistical distribution. There is no assumption about the errors, and the errors are not present in the equations[7]. MLE focuses on finding the parameter values that maximize the likelihood of observing the sample data.

$$\text{MLE} = L(w) = \sum_{i=1}^{n} (p(yi|xi;w))^{yi}(1-p(yi|xi;w))^{1-yi} \tag{2}$$

Here, $L(w)$ is the likelihood function we aim to maximize, n is the total number of samples, yi is the observed binary outcome for the i-th patient, where the outcomes are $(1,0)$, xi is the feature vector for the i-th patient, w is the vector of coefficients (weights) associated with the features, and $p(yi|xi;w)$ is the probability of the outcome yi given the features xi and the coefficient vector w. MLE finds the coefficients w maximizing the likelihood of observing the given outcomes yi in the dataset, fitting the logistic regression model to the data.

2.3 Newton-Raphson's optimization of the Logistic Regression

The Newton-Raphson's optimization method of the Logistic Regression stands as an alternative to Stochastic Gradient Descent and defines the Logistic Regression manually, instead of using scikit-learn's libraries. The method defines the sigmoid function, which is the function modelling the probability that a binary outcome y equals 1, and creates a Hessian matrix with the 2nd derivatives:

$$\frac{d^2C}{d\beta d\beta^T} = H = x^{T(WX)} \tag{3}$$

In this method, we first assume that we have the gradient $g = x^T(P-y) = g(\beta)$ through the probability we have defined. In this case our PDF is a Bernoulli distribution, as we have a binary classification problem, w. possible outcomes 0 and 1, $P(\beta)$. The 2nd derivative, the Hessian depends on β, yielding $H = X^T W(\beta)X = H(\beta)$. We expand $C(\beta)$ around $\hat{\beta}$, the value that minimizes the cost function C, which in our case is the negative log-likelihood function, by using a Taylor series expansion. $\beta^-\beta(m)$ refers to iteration - μ - $\frac{dC}{d\beta} = d\Sigma\beta(C) = 0$. We rewrite this as $C(\beta^{)} = C(\beta^{(}n)))$, where we assume β to be known. Then we Taylor-expand around an unknown value. So we find $(g^{(n)})^T(\beta^-\beta^n)$. Then we get a 2nd derivative. The $0th$ term is just a constant, resulting in the formula

$$\beta - (\beta^n)^T + \frac{1}{2}(\beta^-\beta^{(}n)^T * H(\beta^n)(\beta^-\beta^n) + ... \Leftarrow \frac{d^{2C}}{d\beta d\beta^T}\beta = \beta^n = H(\beta^n) \tag{4}$$

.

$$b = \beta - \beta^{(a)} \Rightarrow C(\hat{\beta}) = C(\beta^{(n)}) + g^{(n)T}b + \frac{1}{2}b^T H^{(n)} b + \ldots \tag{5}$$

Keeping the 2nd order in b only:

$$\frac{dC}{db} = g^{(n)}$$

This is the same as the gradient calculated at the value n, so:

$$g^{(n)} - H^{(n)}b = 0 \Rightarrow b = \beta - \beta^{(n)} \Rightarrow \beta = \beta^{(n)} - (H^{(n)})^{-1}g^{(n)} \tag{6}$$

In this method, to find the parameters minimizing the negative log-likelihood function, Newton-Raphson's method computes the gradient and the regularized Hessian matrix of the negative log-likelihood function for each iteration. If the Hessian is singular, the pseudo-inverse of the Hessian is used instead.

2.4 K-Nearest Neighbour algorithm

K-Nearest Neighbour (kNN) is a non-parametric, instance based learning algorithm which is used for classification tasks. This means kNN cannot be defined by the fixed set of parameters, as those parameters grow with the training data. Because of this, we can consider kNN a ”lazy-learner” [8] as it does not involve a training phase to build a model. Instead, kNN stores the entire training dataset and uses it directly for classification.

To classify a new data point, kNN identifies the k closest neighbours in the training data based on a selected distance metric. It is common to use Euclidean distance, as shown in equation 7, or Manhattan distance, which is shown in equation 8.

$$d_E\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = \sqrt{\sum_k \left| x_k^{(i)} - x_k^{(j)} \right|^2} \tag{7}$$

$$d_M\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = \sum_k \left| x_k^{(i)} - x_k^{(j)} \right| \tag{8}$$

The new data point is then assigned to the class label that is most frequent among these k neighbours. This process is known as majority voting. This is illustrated in figure 2.

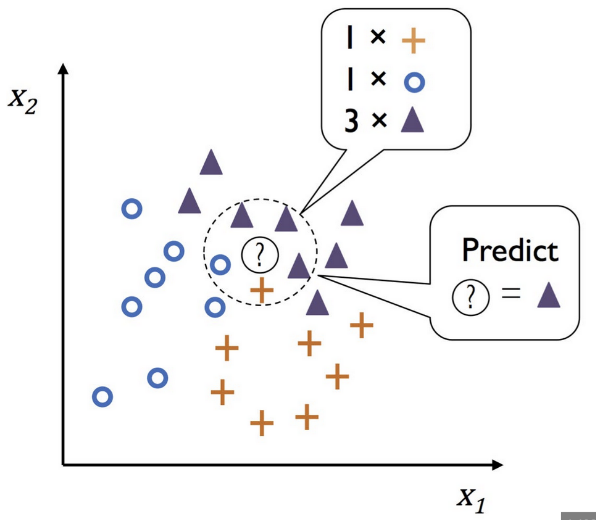


Figure 2: kNN classifier with k=5

The choice of the k-value significantly impacts the effectiveness of the kNN model. For instance, smaller k-values can make the model more sensitive to noise, resulting in high variance. On the other hand, larger k-values tend to smooth out decision boundaries, which can introduce higher bias. Therefore, it is crucial to explore various k-values to identify the optimal one for the model.

2.5 Random Forest algorithm

The Random Forest algorithm is an ensemble learning method primarily used for classification and regression tasks. It operates by constructing a multitude of decision trees during training and outputs the class of the individual trees by using majority voting . Random Forest is known for its robustness, ability to handle large datasets, and effectiveness in reducing overfitting compared to traditional decision trees [9]. The algorithm uses bootstrapping, where random subsets of data are sampled with replacement to train each tree. By averaging or voting on the predictions of these trees, Random Forest reduces the variance seen in single decision trees, leading to a more stable and generalizable model.

The probability of a data point not being selected in a bootstrap sample is approximately 36.8

$$P(\text{not selected}) = \left(1 - \frac{1}{n}\right)^n \approx \frac{1}{e} \approx 0.368 \tag{9}$$

Where n is the total number of data points in the dataset.
These unselected data points are called out-of-bag (OOB) samples, and they can be used to estimate the model's performance without the need for a separate validation set. The OOB error provides an unbiased estimate of the test error, which is particularly useful in evaluating model performance during training [10].

Random Forest also provides a measure of feature importance. This is determined by evaluating how much a feature contributes to reducing the impurity across all trees in the forest. The reduction in impurity, typically measured using the Gini index, or entropy, can be averaged over all trees to provide a ranking of features by their importance. This allows for identifying of the most influential features in the dataset, which can be important for understanding the underlying patterns and improving model performance [10]

The feature importance for a feature is calculated by:

$$I_j = \frac{1}{T} \sum_{t=1}^{T} \sum_{n \in \text{nodes}(t)} \Delta\text{Gini}(n) \cdot 1(\text{feature } j \text{ used in split at node } n) \tag{10}$$

Where Δ Gini(n) represents the decrease in Gini impurity at node n, and the indicator function (feature · j · used) is 1 if feature j is used for splitting at that node.
Once all the decision trees are built, Random Forest aggregates their predictions. For classification tasks, prediction is calculated by majority voting, given by:

$$\hat{y} = \text{mode}(y_1, y_2, \ldots, y_T) \tag{11}$$

Where y1, y2... yt are the predictions from each tree T, and the final prediction $\hat{y}$ is the class with the most votes.

2.6 SVM algorithm

SVM is a supervised machine learning algorithm used for both classification and regression. The SVM algorithm classifies data by finding the optimal line or hyperplane in an N-dimensional space that maximizes the margin between the closest data points of opposite classes.. The dimension of the hyperplane depends upon the number of features. It determines if the hyperplane is a line in 2D space or a plane in an N-dimensional space.It becomes difficult to imagine when the number of features exceeds three. To find the best decision boundary between classes , there is a need to maximize the margin between the points as multiple hyperplanes can be found to classify classe [11].

Though SVM can handle both linear and nonlinear classification tasks. However, to enable linear separation in case of non linear data , kernel functions are used to transform the data higher dimensional space and the choice of kernel function , such as linear kernels, polynomial kernels, Radial basis function kernels or sigmoid kernels depends on data and use case [12].

For a linearly separable dataset, SVM finds a hyperplane that separates the two classes. A hyperplane in an n-dimensional space can be defined as:

$$\mathbf{w}^\top \mathbf{x} + b = 0 \tag{12}$$

Where:

- $\mathbf{w}$ is the weight vector normal to the hyperplane.
- $\mathbf{x}$ is the input feature vector (data point).

- b is the bias or intercept term.

The hyperplane splits the feature space into two halves, each corresponding to one class. Data points are classified based on which side of the hyperplane they fall on [11]. The decision function for SVM decides which side of the hyperplane a new data point lies. It is given by:

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b \tag{13}$$

If $f(\mathbf{x}) > 0$, the data point is classified into one class (e.g., the positive class), and if $f(\mathbf{x}) < 0$, it is classified into the other class (e.g., the negative class).

Support vectors are the data points that are closest to the hyperplane and are crucial for defining the optimal boundary between the classes. The condition for support vectors is:

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) = 1 \tag{14}$$

Where y_i represents the class label of the i-th data point. Support vectors help SVM define the margins of the classifier.

The margin is the distance between the hyperplane and the closest data points (support vectors). The goal of SVM is to maximize this margin, ensuring that the model is robust and generalizes well to unseen data [11]. The margin is given by:

$$\text{Margin} = \frac{2}{\|\mathbf{w}\|} \tag{15}$$

Maximizing the margin helps minimize classification errors on unseen data, as a larger margin provides a clearer distinction between the two classes. SVM is an effective algorithm, especially for classification tasks. By maximizing the margin between classes and allowing some flexibility with slack variables, SVM ensures that the classifier generalizes well to the unseen data. For non-linear data, the kernel trick allows SVM to map data into a higher-dimensional space, making it an even more powerful classifier [12].

2.7 Principal component analysis

Principal Component Analysis (PCA) is used for dimensionality reduction. It transforms a dataset of possibly correlated variables into a set of linearly uncorrelated variables called principal components. The principal components are ordered such that the first few retain most of the variation present in the original dataset. PCA is often used to reduce the number of features while retaining the most important information. This is particularly useful when dealing with high-dimensional data, which can lead to issues like overfitting in machine learning models.It can aslo help in removing multicollinearity (correlation among features) by transforming the original features into a set of uncorrelated components [13]

3 Methods

The application of methods were performed using Python, a programming language widely used for data science and machine learning. We utilized Python version 3.8 and all scripts were executed in a Jupyter Notebook environment. Various Pyhton libraries including Pandas, NumPy, Matplotlib and Scikit-learn were used to develop and evaluate the machine learning models [14].
The steps performed for analysis of the dataset consists of first exploring the dataset, preprocessing the dataset, training four different supervised machine learning models and finally evaluating and tuning the models.

We attempted several configurations on the different models, with a range of parameters and training/test-splits in an attempt to achieve the maximum possible accuracy for each individual method.
For comparing the four different models, we used the same training-test split ratio and number of principal components. This ensures a fair comparison of their performance, allowing us to determine which model performed best on our dataset.
Our conclusion will therefore be based on the results obtained from the different machine learning models using the universal parameters 80/20 training/test and a Principal Component retention of $n = 6$.

3.1 Data exploration

The first step in our analysis involved an exploration of the dataset to gain an understanding of the underlying structure and relationships among the features. Data exploration helps identify patterns, detect anomalies, and guide the steps in the modeling process. Our data exploration included the following key tasks;
Descriptive statistics were computed for each feature in the dataset resulting in measures such as mean, median, standard deviation, and range. This provided an overview of the variability of the features. Next, we examined the dataset for missing values, which could affect the performance of our machine learning models. Histograms and bar plots was plotted to visualize the distributions of the features in the data set.
Furthermore, a correlation heatmap was plotted to examine the relationships between features. This visualization highlights how strongly features are linearly related to one another, with values ranging from -1 to 1. To further explore the data, a 3D scatter plot was generated to visualize the relationship between three key features: age, sex, and chest pain type. The points were colored based on the target variable, indicating the risk of heart attack, 0 or 1.
A principal component analysis was performed on the data with the number of components, n, set to the number of features (13) to create a plot of principal components against the explained variance in the dataset by each principal component. The first three principal components and the feature importance scores were finally plotted, to examine what features that has the biggest impact for the explained variance in the data set.

3.2 Data preprocessing

Before starting training of the different machine learning models, the data was initially separated into the input values with all features (X) and the target values (y). Furthermore, the input values were standardized using scikit-learn's StandardScaler.

The reasoning behind our choice of training/test split is that 70/30 and 80/20 ratios are common practice in machine learning [15]; it balances between training on a substantial amount of data while ensuring a robust evaluation on the test set. The splitting was done by using Scikit-learn's `train_test_split`[16] functionality, with `X` and `y` as inputs. Additionally, `random_state` is used to ensure reproducibility of the results, meaning that `random_state` is an arbitrary number ensuring that the split is the same every time the split is performed.

Principal component analysis was used for the classification models to further reduce dimensionality in the data. The number of principal components, n, were first set to 13 during the data exploration to create a plot of the number of principal components against the explained variance. Then a threshold value of 0.6 on the explained variance was set and 6 principal components were retained.

3.3 Logistic Regression

When applying the MLE, each feature's contribution to the log-likelihood was determined by how much it affects the linear combination $z = \beta_0 + \sum_{j=1}^{m} \beta_j xij$ w. β_j are the model coefficients and xij are the feature values. This contribution was multiplied by the log-likelihood formula. Then we summed these contributions across all samples, accumulating the effects of that feature over the entire dataset. Since the log-likelihood is calculated across all samples, the cumulative contribution was large.

The relative importance of each feature was compared to find what features contributed the most to the total log-likelihood. The results were presented in a diagram, with the features xi on the x-axis, and the cumulative distribution of each feature to the log-likelihood, which is linked to the magnitude of the corresponding coefficient wi and how much that feature impacts the model's predictions, where the coefficients wi that maximize the MLE is what we were aiming to find. Features with higher values on the y-axis were those with stronger relationship with the outcome. Our aim was to maximize the MLE to aid in feature selection, as it allows for identification of the most significant coefficients, revealing critical interactions between risk factors and treatments, allowing for the selection of the most relevant features that contribute the most to predicting heart disease. Another aim of feature selection was to avoid overfitting and simplify the data. As an illustrative application, this can aid in developing pharmaceuticals targeting the most influential variables for heart disease, and by choosing features with high likelihood values, a doctor can assess a patient's risk more accurately and when selecting patients for clinical trials, maximizing MLE for specific features can help identify the individuals most likely to benefit from treatment.

3.4 Optimizing the Logistic Regression

In an attempt to improve the accuracy score of the Logistic Regression, Newton-Raphsons method(described in Section 2.3) was applied to the Hessian matrix of our MLE in an attempt to improve the accuracy. The beta here was the same as $\beta^{(n+1)} = \hat{\beta}$. We performed the iterations, and after n+1 such iterations, we recieved the optimal value. Looking at the expressions explicit for logistic regression: $H^n = H(\beta^n) = x^{T(w(\beta^n))}X$ when setting it up, we found that the gradient was given by $g^n = X^T(P(\beta^n) - y)$. In addition we applied a Ridge coefficient, minimizing the regression coefficients sum of squares

$$\sum_{i=1}^{n}(yi - \beta_0 - \sum_{j=1}^{p}(\beta_j x_{ij})^2 + \lambda \sum_{i=1}^{p}(\beta_j^2) \tag{16}$$

to deal with the possible multicollinearity in the Logistic Regression. This gave us a result of optimized coefficients, and this method of defining the logistic regression method manually instead of using the scikit-learn library was done in an attempt to further improve the accuracy score.

3.5 Feature Selection

To compare with the cumulative contribution to log-likelihood by feature, two model-independent methods, Mutual Information and Chi-Square Test for Feature Selection was applied.

The Chi-Square statistic, measuring the observed frequency vs. the expected frequency for each feature was calculated against the target variable, to see whether the distribution of sample categorical data differed from the expected distribution, where one approach is to use Mutual Information and Log-Likelihood Contribution as initial screening methods, and then applying the Chi-Square Test to refine this selection, which it can be well suited for[17]. The Chi-Square test, given by $X^2 = \sum \frac{(Oi-Ei)^2}{Ei}$, where X^2 is the Chi-Square Statistic, measured the discrepancy between the observed and expected frequencies. Oi are the actual counts observed in each category of the contingency table for the feature and the target variable, and Ei are the counts that would be expected in each category if there were no association between the feature and target. The expected frequency for a cell in a contingency table is $Ei = \frac{Ri * Ci}{N}$, where Ri is the total count of the $i - th$ row, Ci is the total count of the $i - th$ column and N is the no. of observations. Mutual Information on the other hand captured non-linear relationships in addition to the linear dependencies between features and the target, and then gave a feature importance ranking based on how much each feature alone contributed to predicting the target variable, representing a model-independent way of assessing feature importance.

3.6 K-nearest neighbours

To perform k-Nearest Neighbors (kNN) classification, we followed the principles outlined in the theory section 2.4. Before applying the kNN algorithm, we first centered our dataset and then performed min-max normalization to ensure that all features contribute equally to the distance metric. After normalization, the dataset was split into training and test sets. The size of the test-set was set to 20% as in the other models. However, we also use test-set size as a hyperparameter, and explored different

values within the range of 20% to 50% to determine the optimal split. Another key hyperparameter in this algorithm is the number of neighbors (k). In this analysis, we tested k values ranging from 1 to 40 to find the optimal number of neighbors for our classification task.
The analysis was conducted using the Scikit-learn Python library, specifically the KNeighborsClassifier module. In addition to the kNN classification, we also incorporated Principal Component Analysis (PCA) to reduce the dimensionality of the dataset and to examine the model's performance when transformed into a principal component space. We utilized the PCA module from Scikit-learns sklearn.decomposition package to transform the original dataset into a principal component space.

For consistency with other models, we retained 6 principal components and subsequently split the transformed dataset into training and test sets, with the test set again comprising 20% of the data. Next, we plotted the accuracy as a function of k-values ranging from 1 to 40, comparing results with and without PCA transformation. Accuracy plots were generated for both the training and test sets. We used the accuracy, recall, precision and f1 score as our evaluation metric. The predicted values were compared to the actual observed values to assess the performance of the classifier.
To further refine the model, we compared the impact of applying PCA on the models accuracy. Based on these findings, we selected the approach (with or without PCA) that yielded better results and conducted a grid search across varying test set sizes and k-values to find the most optimal combination. Additionally, we evaluated whether the Manhattan or Euclidean distance metric provided higher accuracy, selecting the most effective metric for our final model.

3.7 Random forest

In this study, Random Forest was implemented using the `RandomForestClassifier` from the Scikit-learn library in Python [ref scikit]. The data set was first split in a 70/30 training-test set. After inspection of the accuracy score, the split was then adjusted from a 70/30 split to a 80/20 split for our comparison study, as it seemed to improve most of the models accuracy score.

Furthermore, the data was preprocessed in the similar way as for the previous models, that is, standarization of the data, 80/20 train-test split and with PCA with 6 principal components to reduce the dimensionality of the data. The accuracy score was then calculated again. To further improve the score, we performed hyperparameter tuning. The most optimal parameters were found by performing a grid search with `GridSearchCV` from the Scikit-learn library.

By tuning the hyperparametersnumber of trees (`n_estimators`), maximum depth (`max_depth`), minimum samples required to split a node (`min_samples_split`), minimum samples required at each leaf node (`min_samples_leaf`), and number of features considered at each split (`max_features`)we optimized the model to balance between accuracy and overfitting.
The final parameters used in the random forest model were set to be:

`max_depth=10`, `max_features='sqrt'`, `min_samples_leaf=4`, `min_samples_split=2`, `n_estimators=100`, and finally `random_state=42`.

After tuning, the random forest model was trained again, before finally calculating the accuracy scores and generating a confusion matrix to visualize the performance of the final model.

3.8 SVM

After the data preprocessing, two dataframes with the selected PCA'S were created; one for the test set and one for the training set. Furthermore, a SVM model was then initialzed using scikit-learns SVM library [14]. The defined parameter were the kernel type, set to 'rbf', probability set to true and the random state set to 42.
After calculating the accuracy score, we performed hyperparameter tuning by using a gridsearch. This was also done using the GridSearch library from scikit-learn. A pipeline was created for the second round of training, with the parameteres StandardScaler, PCA with n=6 components, and SVM with kernel='rbf', probability=True, random state=42. Next, a cross validation was performed to accsess the mean accuracy scores.
The grid search included parameteres for the pipeline, and parameters for the SVM model. After finding the most optimal parameters, a new SVM model was trained with these parameteres, and the final accuracy score was then calculated.

4 Results and Discussion

4.1 Data exploration

The following visualisations were generated during the data exploration, shown in figure 3, 4, 5, 6, and 7.

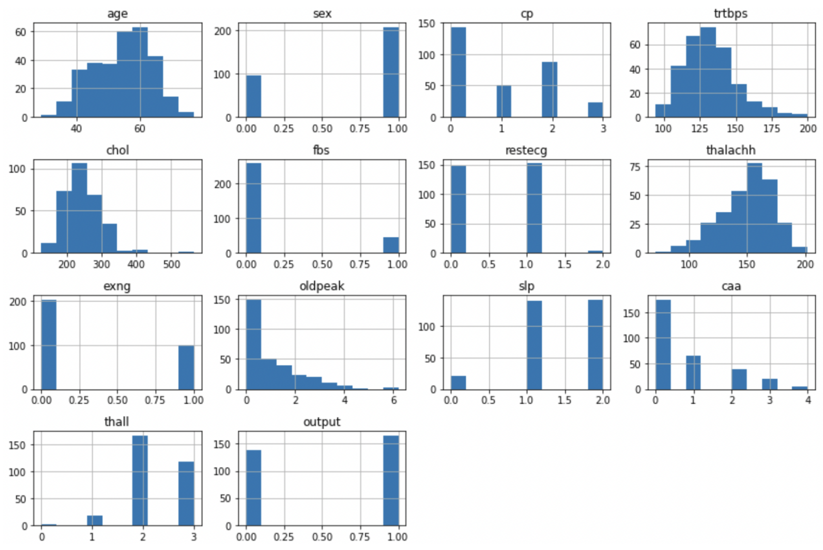


Figure 3: **Histogram showing the distribution of all features in the data set.**

The histograms shows the distribution in the different features in the data set, and also possible outliers. It seems like there are twice as many male patients in this data set, compared to the amount of females. There is also many older patients aged 60 and above, compared to younger patients. It also seems like there are many patients getting the output value 1, indicating a high risk for heart attack.

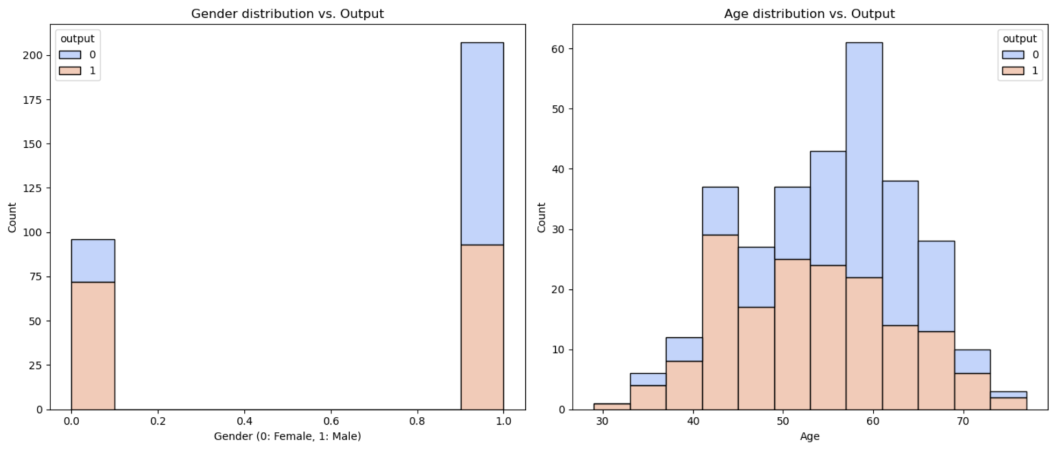


Figure 4: **Stacked barplots. The left plot displays the gender distributions and the output value, 1 (red) indicating high risk of heart attack and 0 (blue) indicating not high risk of heart attack. The right plot shows the age distribution and the output value, 1 or 0. The x-axis shows either age or gender, while the y-axis shows the count.**

The bar plots in 4 shows that most females in this data set has higher risk of getting heart attacks, while there also is few females in the data set compared to males. The count of males in this data set is approximately twice as many as females. This indicates that the females are more likely to get

heart attack, according to this dataset. However, there is almost the same amount of patients getting the output value 1, high risk of heart attack, across both genders.
The barplot to the right shows the age distrubutions in this data set and the output value. There are more patients between 40 ang 50 years old with high risk of developing heart attack.

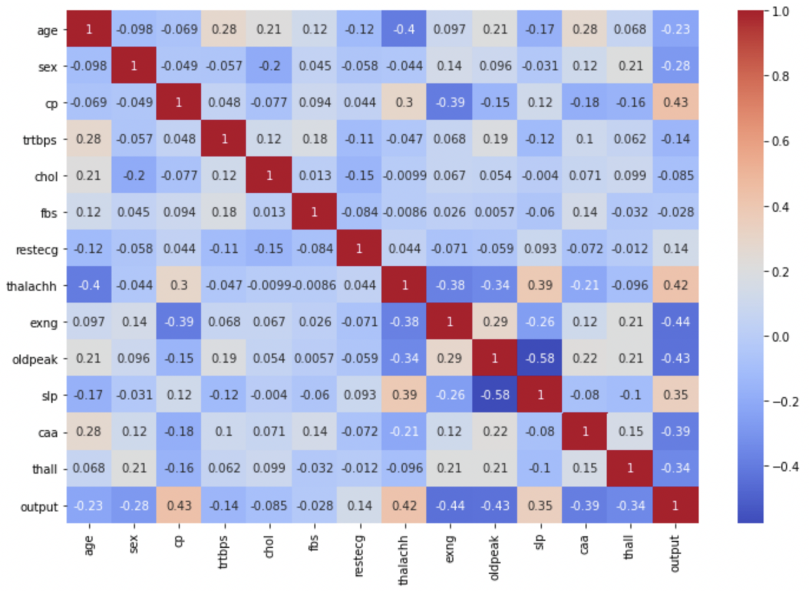


Figure 5: **Correlation heatmap showing the correlation values between all features in the data set, with the feature names on both axes.**

Figure 5 shows how the different features are correlated to each other, and to the output. The features named cp, thalachh and slp seem to have the highest positive correlation to the output. This indicates that if cp (chest pain) increases, the output value, risk for developing heart attack, also increases. On the other hand, exng, oldpeak, caa and thall seem to have the highest negative correlation with the output. This indicates that if any of these features, for example exng (Excercise-Induced Angina) increases, then the output value will decrease.

Figure 6 gives a representation of the relationship between the features age, sex, chest pain type and the output. The plot further proves that there are more females that has a high chance of heart attack (red dots), compared to males with more blue dots. We can also tell higher chest pain type (the z-axis) has more red dots, indicating that higher chest pain might lead to higher risks for heart attack. There is also more older patients having a higher chance for heart attack, especially for the females.

The plots shown in figure 7 generated after performing PCA with all 13 features, show how many principal components that explains the variance in the data set, singular values and the cumulative plot of principal components. From the first plot to the far right and far left, we concluded that 6 principal components would suffice in reducing the dimensionality of the data set, as these would have the highest factors in explaining the data sets variability. Furhtermore, we also made a plot of the first three principal components and the feature importances for each feature in the data set, shown in figure 8.
The left plot shows principal component 1 and how the feature oldpeak has the highest positive feature importance. The feature oldpeak contains numbers from 0-6, representing values from an electrocardiogram indicating heart illness. The feature named thalachh has the highest negative feature importance. The feature thalachh contains values that determines the heart health by measuring cardiac response to exercise. For higher values of thalacc, the output value decreases. Moreover, the feature trbps has the highest positive feature importance for the second principal component, while sex is the feature with highest negative importance. For the third principal component, sex has the

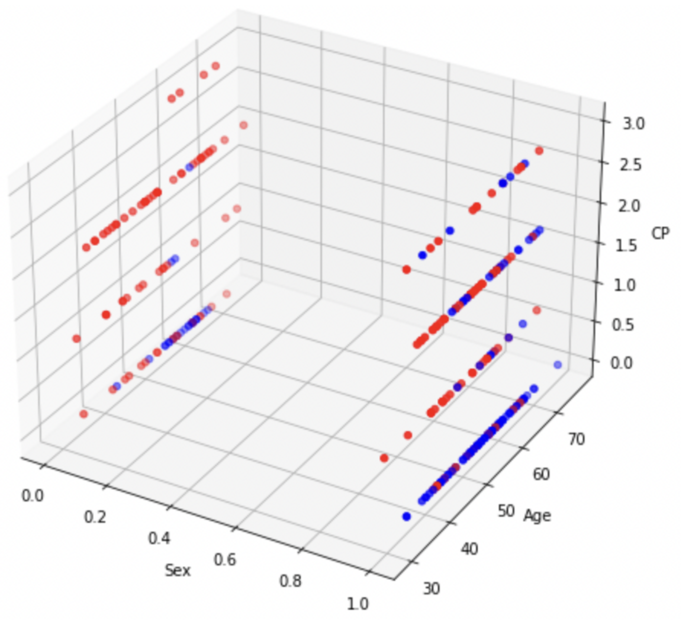


Figure 6: **3D-scatterplot with the sex (0 for female, 1 for male) on the x-axis, age on y-axis and the chest pain type on the z-axis. The dots are colored after the output value for the risk of heart attack, blue being 0 and red being 1.**

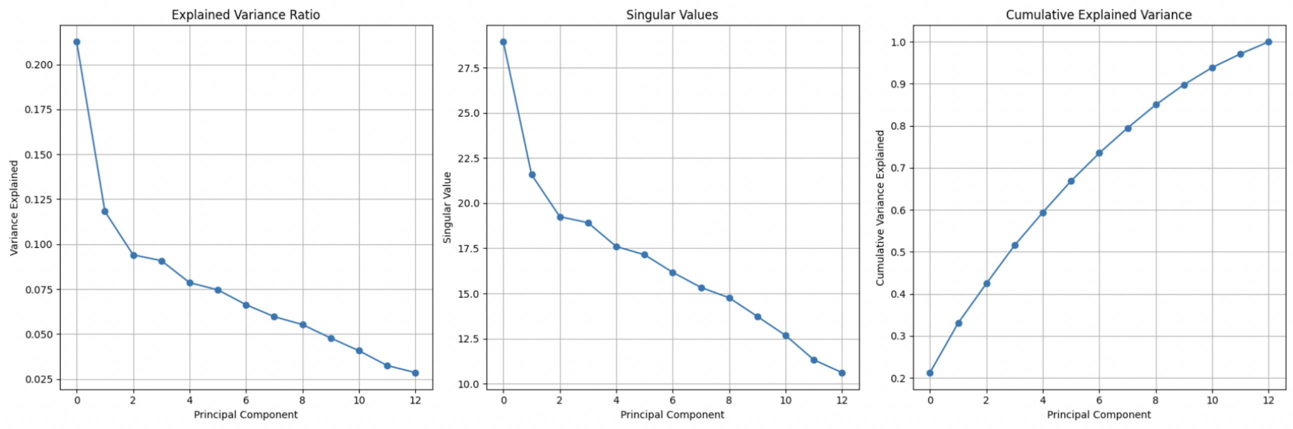


Figure 7: **Far left: plot of the principal components on the x-axis against the percentage of explained variance in the data set on the y-axis. Middle: The principal components on the x-axis plotted against he singular values on the y-axis. Far right: The cumulative plot of the principal components on the x-axis against the explained variance in the data set on the y-axis.**

highest positive feature importance, while chol has the highest negative feature importance.

4.2 Supervised learning

All learning methods were instances of supervised learning, except PCA which is an instance of unsupervised learning. All of the supervised learning methods followed the same 80/20 test/training split and a principal component retention of $n = 6$ in the following comparison study.

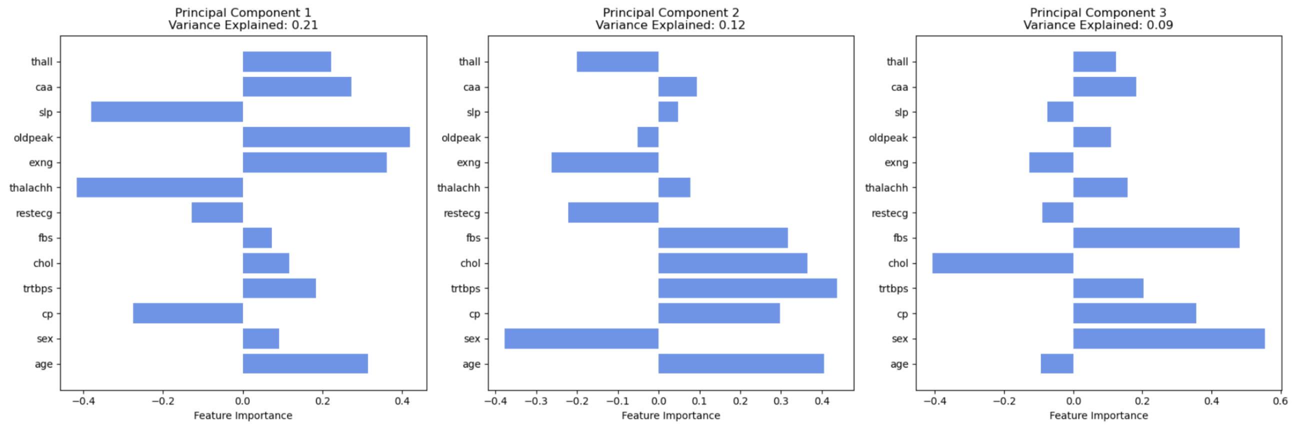


Figure 8: **Far left: plot of the first principal components feature importances. the x-axis shows the feature importances, while the y-axis shows the feature names. Middle: the feature importances for the second principal components. Far right: the feature importances for the third principal components.**

4.2.1 Logistic regression

Logistic Regression with the universal parameters 80/20 and $n = 6$ gave an accuracy score of 0.89, a better accuracy score than the accuracy score of 0.85 obtained without doing a PCA retention of $n = 6$. In an attempt to improve on the accuracy obtained from the logistic regression, the optimization using the Hessian matrix and Newton-Raphson's method was applied along with a Ridge regression coefficient. The optimized Logistic Regression returned the accuracy score of 0.87, but likely performed better at classifying the target variable correctly, and the convergence speed is indicative of the efficiency of the optimization algorithm, suggesting it was very efficient. Elastic Nets filtering method was applied in addition to this, giving the accuracy score of 0.89 for any combination of Ridge and Lasso coefficients in the range $(0, 1)$.

In the process of trying to select the most important features contributing to the binary classification result, the cumulative feature contributions to the total log-likelihood computing in the Logistic Regression[Figure 9] was used as a filtering method. Two other filtering methods, the Chi-Square Test,

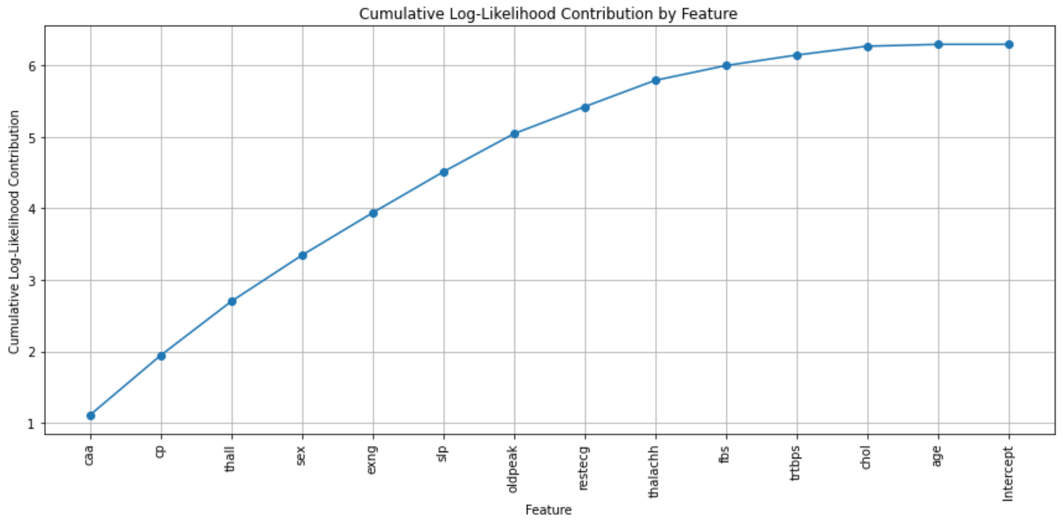
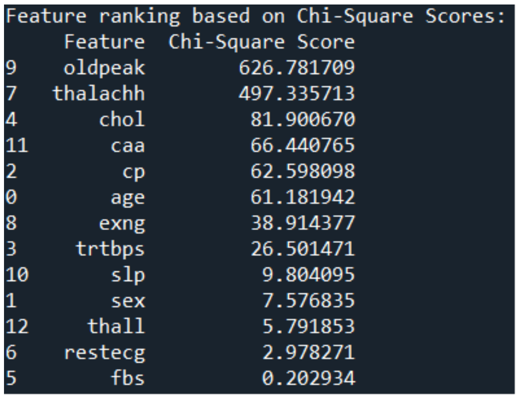


Figure 9: **The distribution of Cumulative Log-likelihood contribution by feature.**

shown in figure 10.

and the model-independent Mutual Information [Figure 11]



```
Feature ranking based on Chi-Square Scores:
     Feature  Chi-Square Score
9    oldpeak        626.781709
7   thalachh        497.335713
4       chol         81.900670
11       caa         66.440765
2         cp         62.598098
0        age         61.181942
8       exng         38.914377
3     trtbps         26.501471
10       slp          9.804095
1        sex          7.576835
12     thall          5.791853
6    restecg          2.978271
5        fbs          0.202934
```

Figure 10: **The Features with the highest Chi-Square Test score.**

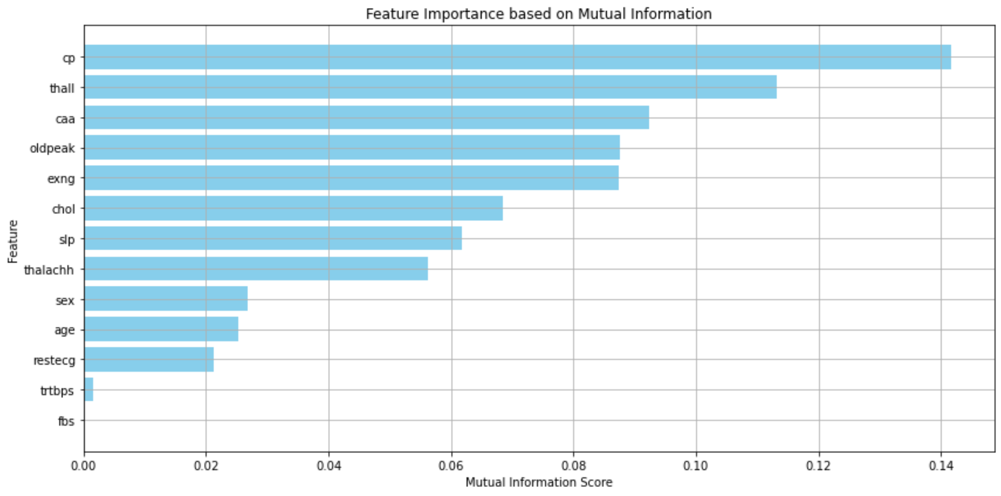


Figure 11: **Feature Importance based on the Mutual Information filtering method.**

selected different features than the model-dependent cumulative feature contributions, which was expected as the cumulative log-likelihood contribution assesses the cumulative importance of features based on their contribution to optimizing the logistic regression as a sum of the contribution over all samples, whereas the Mutual Information method tries to capture relationships in the statistical data but does not correlate with the Logistic Regression model's predictions.
This points to some of the difference being explained by the fact that one of the methods measures a cumulative effect whereas the others do not, and the fact that the cumulative contribution measurement measures its effect on optimizing the log-likelihood, which is a measurement specific to the Logistic Regression model, whereas the other feature selection methods do not rely on a model. The features "thalachh", "oldpeak", "chol" robustly stood out as features that scored high across the three feature selection methods, whereas "caa", "thall" and "cp" scored the lowest in the Cumulative Log-Likelihood Contribution method, but highest in the Mutual Information method. The difference in the highest and lowest values for the features was significantly lower for Mutual Information and Cumulative Log-Likelihood than for the Chi-Square Test, where the scores had very high variation, which is expected as the Chi-Square Test is sensitive to the distribution of data and will have high variance when some features have very small samples. On the basis of the comparison of the features selected by these three methods, we decided not to discard any features based on these filtering methods due to the different nature of what they explain.

4.2.2 K-nearest neighbours

After splitting the dataset into a training set (80%) and a test set (20%), and applying Principal Component Analysis (PCA) to reduce the dimensionality to 6 principal components, we achieved an accuracy of 86% on the test set and 79% on the training set. For this analysis, we used the Manhattan distance metric and selected a k-value of 36. To ensure the selection of the optimal k-value, we tested the model with k-values ranging from 1 to 40, as described in the method section. Figure 12 shows the accuracy as a function of different k-values, including a comparison with the kNN model without PCA. This comparison helps to highlight the effect of dimensionality reduction on model performance.

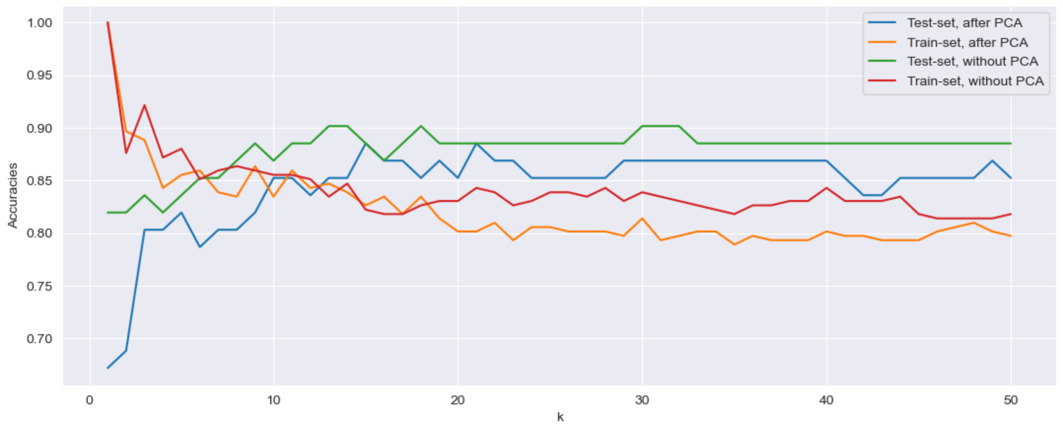


Figure 12: Plot showing accuracy as the function of k-values. Here we can clearly see that kNN model without PCA performs best, giving the highest accuracy score on test-set with 90% for $k = 13$, while the highest accuracy score for kNN model with PCA gives 88% for $k = 15$

To further improve the model, we treated the test set size as a hyperparameter and performed a grid search to find the optimal combination of test set size and k-value. Figure 13 presents a heatmap showing the results of various combinations of k-values (ranging from 1 to 40) and test set sizes (ranging from 10% to 55%).

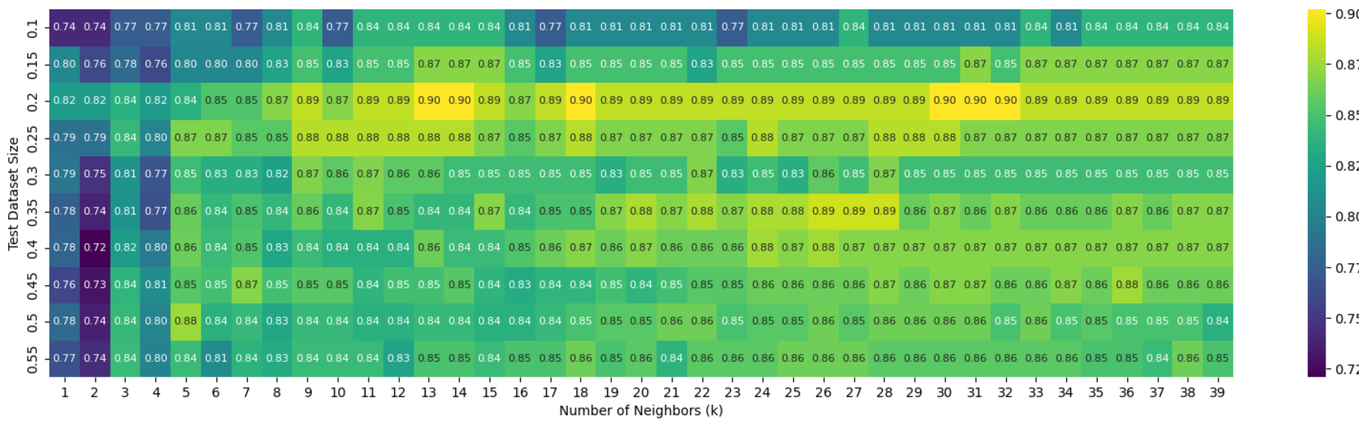


Figure 13: Grid search for test-set size and k-values. Which shows that the optimal combination of those two hyperparameters are test-set size 20% and $k = 35$, $k = 13$, $k = 14$ and $k = 18$

We found that the most optimal kNN model achieved 90% accuracy on the test set and 82% on the training set. This was obtained using $k = 35$, without PCA, a test size of 20% and the Manhattan distance metric. Further evaluation is provided in the table 1, and the confusion matrix in figure 14

The results presented shows a clear distinction in accuracy when applying PCA versus not applying it. It seems that the accuracy gets worse. While PCA is typically beneficial for reducing dimensionality and improving computational efficiency, it seems that it may not always lead to a better performance, especially in distance-based algorithm like kNN. Similar results hsve been observed in text classifica-

Metric	Score
Accuracy (Test-set)	86.9%
Accuracy (Train-set)	78.9%
Precision	87.0%
Recall	86.9%
F1 Score	88.5%

Table 1: Evaluation metrics for the kNN model

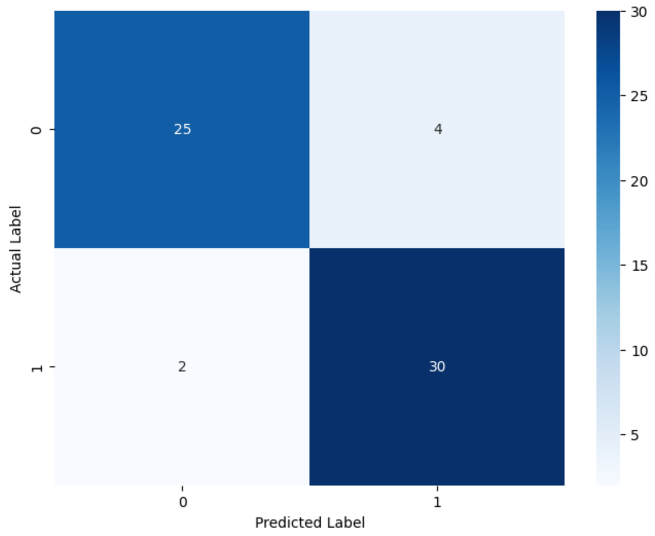


Figure 14: Confusion matrix for kNN algorithm showing number of misclassified and correctly classified samples. Here we have 6 misclassified classes, and 55 correctly classified classes.

tion tasks [18], where PCA imporved performance with some classifiers, but negatively affexted kNN performance. PCA reduces dimensionality by emphasizing variance, but in doing so, it may discard features that are important for calculating accurate distances between data points. This can distort the relationships that kNN depends on for classification. By contrast, the model without PCA retains all the original feature information, leading to better performance. Thus, in this case, PCA oversimplified the feature space, negatively affecting kNNs accuracy.

4.2.3 Random Forest

In the first round of training, the accuracy score for the model resulted into 0.81 for the test set, and 1.00 for the training set. This indicated that there is overfitting on the training data. To decrease the overfitting, we performed hyperparameter tuning by using a grid search to find the best parameters. This resulted into the final accuracy score of 0.84 for the test set, and 0.93 for the training data. A confusion matrix was then generated to visualize the amount of misclassifications and ultimately the performance of the model, shown in figure 15.

By performing hyperparameter tuning, the gap between the performance on test and training data decreased, but there is still room for improvement. A difference of 0.9 indicated that there is still some overfitting on the training data. The final model misclassified 10 samples, and this might be due to the multicolinarities within the features in the data set, which can result into difficulties in separating samples and successfully classifying the samples into the correct class. These issues could have been solved by trying to optimize the model as much as possible, to handle to overfitting and multicolinearities in the data set. Methods such as feature extraction, adjusting the train-test split, and others could have improved the accuracy. However, this was not done as the objective of this study is to compare four models that has been preprocessed in the same way to compare the performances of each model.

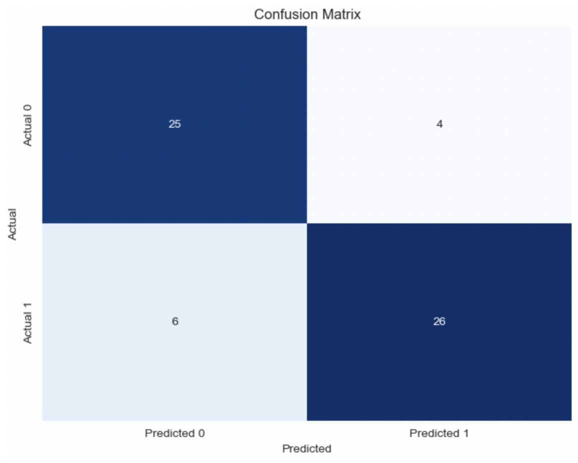
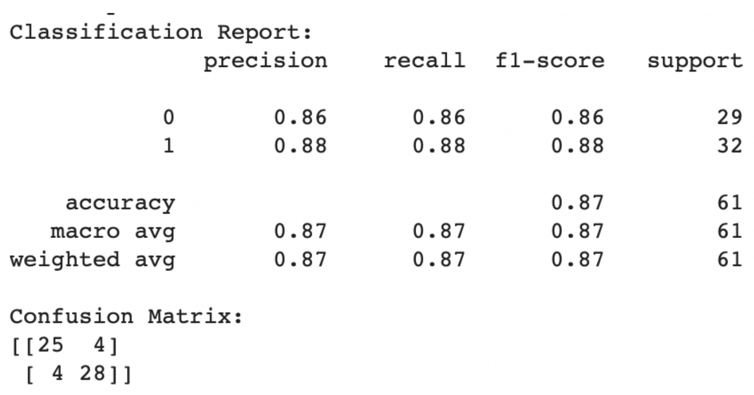


Figure 15: **Confusion matrix showing the amount of misclassified and correctly classified samples for the random forest model. the x-axis shows the predicted values, while the y-axis shows the actual values.**

4.3 SVM

After training the Support Vector Machine (SVM) model on the PCA-transformed training data, the model was evaluated using the test data. The results of this evaluation were summarized through several key metrics, including accuracy, precision, recall, F1-score, and a confusion matrix, shown in figure 16.



```
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.86      0.86        29
           1       0.88      0.88      0.88        32

    accuracy                           0.87        61
   macro avg       0.87      0.87      0.87        61
weighted avg       0.87      0.87      0.87        61

Confusion Matrix:
[[25  4]
 [ 4 28]]
```

Figure 16: **Classification report for the SVM model, and confusion matrix in the bottom.**

The model correctly predicted 86.89% of the test instances, which indicates a high level of accuracy. This suggests that the model performs well in distinguishing between the two classes.
Recall (also known as sensitivity or true positive rate) is the proportion of actual positive instances that were correctly identified by the model. The model correctly identified 86% of the actual class 0 instances and 88% of the actual class 1 instances
An F1-score of 0.86 for class 0 indicates a good balance between precision and recall, while an F1-score of 0.88 for class 1 indicates a similarly good balance between precision and recall.
The confusion matrix shows how the model had only 8 misclassifications out of 61 instances which indicates a low error rate.
After hyperparameter tuning, the model did not seem to improve the accuracy score, in fact, none of the metrics changed. Methods that can be done to improve the accuracy score is extending the parameters in the grid search, adjusting the train-test split, preprocessing the data differently such as changing the number of principal components, and also cleaning the data by feature extraction and scaling.

4.4 Comparison of Results with Supervised Learning Methods

A comparison of all accuracy scores is shown in figure 17 and table 2.

Model	Accuracy
Logistic Regression	89%
K-Nearest Neighbors (kNN)	87%
Support Vector Machine (SVM)	87%
Random Forest	84%

Table 2: Comparison of accuracy for all four models where the data was splitted in 80/20 test-train sets and had PCA with 6 principal components

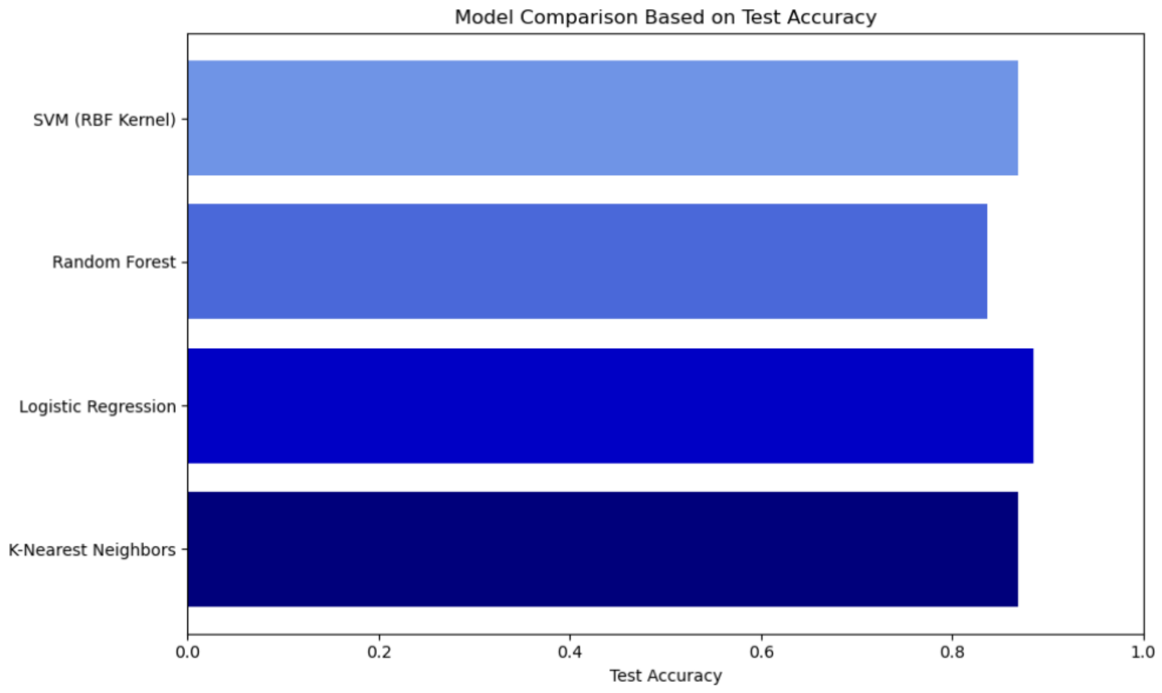


Figure 17: **Comparison of all accuracy scores from each model. the x-axis**

This figure displays how the four models performed similarly, with an accuracy score between 0.8 and 0.9. The model with the highest accuracy score was the logistic regression, with an accuracy score of 0.89. The model with the lowest accuracy score was the random forest model with an accuracy score of 0.84. The models also performed as expected according to the data set publishers, as they have posted the baseline model accuracy scores for most machine learning models, with the logistic regression model scoring 0.815, the random forest model scoring 0.802 and SVM model scoring 0.657 [5]. Our models performed better which might be due to differences in preprocessing the data and tuning the model.

4.5 Sources of error

A test for multicollinearity was performed on the original dataset, using the Variance Inflation Factor (VIF), and identified five features with a VIF above 10. Although removing features with high VIF is commonly done to reduce multicollinearity, this approach does not necessarily lead to higher accuracy. In the context of our project, this is due to several factors. Ridge regularization adds a penalty term to the negative log-likelihood function proportional to the square of the magnitude of the coefficients. This penalty stabilizes the Hessian matrix, making its inversion more reliable. However, removing high-VIF features alters the feature set, which changes the size and structure of the Hessian matrix. Consequently, the regularization that was effective for the original feature set may not be as effective

for the new set, potentially impacting the convergence and stability of the Newton-Raphson method. Moreover, while removing high-VIF features may reduce noise, it could also lead to the loss of valuable information. Multicollinearity is effectively addressed by PCA since the principal components are linearly independent. This means that PCA not only eliminates multicollinearity but also reduces dimensionality, allowing the model to focus on the most significant components of the data. For the logistic regression, the initial accuracy of 0.85, and the increase to 0.89, could be attributed to both the removal of multicollinearity and the dimensionality reduction achieved through PCA. By contrast, removing the high-VIF features from the original dataset prior to applying splitting and PCA resulted in a lower accuracy score of 0.84, while applying Newton-Raphson's returned an even lower accuracy of 0.82.
This suggest that while multicollinearity was adressed, the reduced dataset lost valuable information, and the altered Hessian matrix negatively impacted the model's performance, leading to a 1 percent and 3 percent drop in accuracy, respectively.

By not removing any features in the data set, you can hinder losing important data for prediction. On the other hand, including irrelevant or redundant features in the model can dilute the predictive power of useful features and confuse the model. This is something that needs to be balanced if one decides to remove features from the data set, and in our case we did not remove any features. Another source of error is not performing a grid search that is extensive enough to find the best parameters for the models, and also possible outliers in the data set that creates noise in the data.

5 Conclusion

This project evaluated four machine learning models, which were the Logistic Regression, K-nearest Neighbours, Random Forest, and Support Vector MAchine. This was done to predict heart disease based on a dateset of 13 features and 303 samples. All models were trained and tested using an 80/20 train-test split, and dimensionality reduction was performed using Principal Component Analysis to reduce multicollinearity and improve performance. Hyperparameter tuning, cross-validation and feature selection techniques were also applied to optimize the models.

Logistic regression likely performed and best due to the linearity of the PCA-reduced dataset, the low dimensionality and the small size of the dataset, and its probabilistic approach in dealing with binary classification tasks. It is computationally inexpensive and converges quickly, and has fewer parameters to tune than kNN, SVM and Random Forest. This simplicity makes Logistic Regression avoid overfitting.

The k-Nearest Neighbours model achieved an accuracy of 87% after optimizing the number of neigbours ($k = 35$) and using the Manhattan distance metric. The performance of kNN was slightly worse when PCA was applied, likely due to the model's way of colculating distance metrics, which PCA might have distorted.

Random Forest, known for its robustness and ability to handle large datasets, had the lowest accuracy at 84%. Despite tuning parameters like the number of trees, macimum depth, and minimum samples, it still showed signs of overfitting, as indicated by a significant gap between the training and test accuracy.

SVM, using a radial basis function (RBF) kernel, also achieved an accuracy of 87%. The use of kernel methods allowed SVM to handle non-linear relationships between features, but the models performance did not significantly improve after hyperparameter tuning.
Overall, Logistic Regression emerged as the most efficient and accurate model. Table **??** shows the overview of the all accuracy score for all four models.

5.1 Future Work

In the future we aim to work on more complex datasets, re-evaluating the computational cost and ability to get accurate results using the different supervised learning methods and filtering methods.

References

[1] Cardiovascular Diseases[Internet]. WHO; 2024 [retrieved August 25,2024]. Available from:
https://www.who.int/health-topics/cardiovascular-diseases#tab=tab_1

[2] Global Heart and Circulatory Diseases Factsheet[Internet]. British Heart Foundation; 2024 [retrieved August 28,2024]. Available from:
https://www.bhf.org.uk/-/media/files/for-professionals/research/heart-statistics/bhf-cvd-statistics-global-factsheet.pdf?rev=e61c05db17e9439a8c2e4720f6ca0a19&hash=6350DE1B2A19D939431D876311077C7B

[3] Xu Y., Goodacre R. On Splitting Training and Validation Set: A Comparative Study of Cross-Validation, Bootstrap and Systematic Sampling for Estimating the Generalization Performance of Supervised Learning[Internet]. Liverpool: University of Liverpool; 2018 [retrieved August 14,2024]. Available from:
https://www.researchgate.net/publication/328589123_On_Splitting_Training_and_Validation_Set_A_Comparative_Study_of_Cross-Validation_Bootstrap_and_Systematic_Sampling_for_Estimating_the_Generalization_Performance_of_Supervised_Learning

[4] Rahman R., Heart Attack Analysis & Prediction Dataset[Internet]. Kaggle: 2021 [retrieved August 13, 2024]. Available from:
https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset/data

[5] Janosi A., Steinbrunn W., Pfisterer M., Detrano R., Heart Disease [Internet]. UC Irvine: Cleveland, Hungary, Switzerland, VA Long Beach; Data donated in 1988 [retrieved August 13, 2024]. Available from:
https://archive.ics.uci.edu/dataset/45/heart+disease

[6] Senaviratna, N.A.M.R., Cooray, T.M.J.A., Diagnosing Multicollinearity of Logistic Regression Model[Internet]. Sri Lanka: Asian Journal of Probability and Statistics; 2019 [retrieved August 16, 2024]. Available from:
http://library.go4manusub.com/id/eprint/60/1/Senaviratna522019AJPAS51693.pdf

[7] Perraillon M.C., Maximum Likelihood Estimation[Internet]. Colorado: Anschutz Medical Campus University of Colorado; 2020 [retrieved August 17, 2024]. Available from:
https://clas.ucdenver.edu/marcelo-perraillon/sites/default/files/attached-files/week_6_mle_perraillon_0.pdf

[8] Zhang M., Zhou Z., ML-KNN: A Lazy Learning Approach to Multi-Label Learning[Internet]. Hong Kong: International Conference on Machine Learning; 2007 [retrieved August 29, 2024]. Available from:
https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel/MLKNN.pdf

[9] Random Forest algorithm in Machine Learning[Internet]. GeeksforGeeks: Uttar Pradesh; 2024[retrieved August 29,2024]. Available from:
https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/

[10] Donges N., Random Forest: A Complete Guide for Machine Learning [Internet]. Builtin: Chicago; 2024 [retrieved August 29,2024]. Available from:
https://builtin.com/data-science/random-forest-algorithm

[11] Support Vector Machine (SVM) Algorithm [Internet]. GeeksforGeeks: 04.06.2024 [retrivied 03.09.2024]. Available from:
https://www.geeksforgeeks.org/support-vector-machine-algorithm/

[12] Gandhi R. Support Vector Machine Introduction to Machine Learning Algorithms [Internet]. 07.06.2028 [retrievied 03.09.2024] Available from:
https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[13] Principal Component Analysis(PCA) [Internet]. GeeksforGeeks: 26.07.2024 [retrivied 03.09.2024]. Available from:
https://www.geeksforgeeks.org/principal-component-analysis-pca/

[14] scikit-learn-developers, scikit-learn Machine Learning in Python [Internet]. Scikit-learn; 2024[retrieved August 27,2024]. Available from:
https://scikit-learn.org/stable/

[15] Gholamy A., Kreinovich V., Kosheleva O., Why 70/30 or 80/20 Relation Between Training and Test Sets: A Pedagogical Explanation[Internet]. Texas: University of Texas at El Paso; 2018[retrieved August 15,2024]. Available from:
https://scholarworks.utep.edu/cs_techrep/1209/

[16] scikit-learn-developers, Train-test-split[Internet]. Scikit-learn; 2024[retrieved August 14,2024]. Available from:
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[17] Zhai Y., Song W., Liu X., Liu L., A Chi-Square Statistics Based Feature Selection Method in Text Classification[Internet]. Beijing: International Conference on Software Engineering and Service Science; 2018 [retrieved August 22, 2024]. Available from: https://www.researchgate.net/publication/331850396_A_Chi-Square_Statistics_Based_Feature_Selection_Method_in_Text_Classification

[18] Taloba A.I., Eisa D.A., Ismail S.S.I., A Comparative Study on using Principal Component Analysis with Different Text Classifiers[Internet]. arXiv; 2018 [retrieved August 29, 2024]. Available from:
https://arxiv.org/abs/1807.03283

6 Appendix

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from mpl_toolkits.mplot3d import Axes3D
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
import numpy as np
from scipy.special import expit


#%% Loading dataset
data = pd.read_csv('heart.csv', low_memory=False)

# copy of data
raw_df = data.copy()

# Defining input data and the target values
X = raw_df.drop(columns=[raw_df.columns[-1]])
y = raw_df[raw_df.columns[-1]]


#%% Raw data exploration

# Descriptive statistics
descr_stats= raw_df.describe()
print(descr_stats)

# Searching for missing values
print("Missing values in dataset:\n", raw_df.isnull().sum())
# no missing values in the dataset.

# Histogram for features in the dataset
raw_df.hist(figsize=(12, 8))
plt.tight_layout()
plt.show()

# Correlation heatmap
corrmat = raw_df.corr()
top_corr = corrmat.index
plt.figure(figsize=(12,8))
heatmap = sns.heatmap(raw_df[top_corr].corr(),annot=True,cmap="coolwarm")

# 3D Scatterplot
# For features age, sex and cp (chestpain type), colored by output.
# Sex = 0 indicates female,
# Sex = 1 indicates male.
# Cp have values 0, 1, 2 or 3. 0 being typical angina, 3 being no symptoms.
# Output = 0 means lesser chance of heartattack, colored blue.
# Output = 1 means higher chance for heart attack, colored red.

fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
x = raw_df['sex']
y2 = raw_df['age']
z = raw_df['cp']
c = raw_df['output']

# Defining colors of the dots
colors = np.where(c == 0, 'blue', 'red')

# Scatter plot with color based on 'output'
ax.scatter(x, y2, z, c=colors)
ax.set_xlabel('Sex')
ax.set_ylabel('Age')
ax.set_zlabel('CP')
plt.show()


# plot of distrubution of genders and the output in dataset
def plot_gender_vs_output(data):
    fig, ax = plt.subplots(1, 2, figsize=(14, 6))

    sns.histplot(data=data, x="sex", hue="output", multiple="stack", ax=ax[0], palette="coolwarm",
    kde=False)
    ax[0].set_title("Gender distribution vs. Output")
    ax[0].set_xlabel("Gender (0: Female, 1: Male)")
    ax[0].set_ylabel("Count")

    sns.histplot(data=data, x="age", hue="output", multiple="stack", ax=ax[1], palette="coolwarm",
    kde=False)
    ax[1].set_title("Age distribution vs. Output")
    ax[1].set_xlabel("Age")
    ax[1].set_ylabel("Count")

    plt.tight_layout()
    plt.show()

plot_gender_vs_output(raw_df)



#%% PCA of scaled data to explore data:
# Standardizing the raw data using StandardScaler
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Performing PCA and fitting on data
```

```
pca = PCA(n_components=13)
pca.fit(X_scaled)

loadings = pd.DataFrame(pca.components_.T,
columns=['PC%s' % _ for _ in range(len(X.columns))],
index=X.columns)

# Plot of principal components and explained variance
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 6))
ax1.grid(True)
ax1.plot(pca.explained_variance_ratio_, marker='o')
ax1.set_ylabel('Explained Variance')
ax1.set_xlabel('Principal Components')
ax1.grid(True)
ax1.set_title('Explained Variance in % by principal components')

# Singular values plot
ax2.grid(True)
ax2.set_title("Singular Values")
ax2.plot(pca.singular_values_, marker='o')
ax2.set_xlabel('Principal Component')
ax2.set_ylabel('Singular Value')

# Cumulative plot
variance = np.cumsum(pca.explained_variance_ratio_)
ax3.grid(True)
ax3.plot(variance, marker='o')
ax3.set_ylabel('Cumulative Explained Variance')
ax3.set_xlabel('Principal Components')
ax3.grid(True)
ax3.set_title('Cumulative plot for explained variance')

plt.tight_layout()
plt.show()


#%%
# Plot of pca components against feature importance
def plot_pca_components(pca, X_scaled):
    explained_variance = pca.explained_variance_ratio_
    components = pca.components_

    # Plotting the first three principal components:
    fig, axes = plt.subplots(1, 3, figsize=(18, 6))
    for i in range(3):
        ax = axes[i]
        ax.barh(X.columns, components[i], color='cornflowerblue')
        ax.set_title(f'Principal Component {i+1}\nVariance Explained: {explained_variance[i]:.2f}')
        ax.set_xlabel('Feature Importance')

    plt.tight_layout()
    plt.show()

plot_pca_components(pca,X_scaled)
```

```
#%% Data Preprocessing
# Splitting the data into training and testing sets (80% train, 20% test):
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

n_components = 6
# PCA to pre-process the data
pca = PCA(n_components=n_components)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)



#%% Modeling

# Random Forest
forest = RandomForestClassifier(random_state=42)

# Grid search for finding the best parameteres
rf_params = {
    'n_estimators': [100, 300, 700],  # Number of trees
    'max_depth': [10, 20, 30],             # Maximum depth of each tree
    'min_samples_split': [2, 5, 10],       # Minimum samples required to split a node
    'min_samples_leaf': [1, 2, 4],         # Minimum samples required at each leaf node
    'max_features': ['sqrt', 'log2']       # Number of features to consider at each split
}

gs_random = GridSearchCV(estimator=forest, param_grid=rf_params, cv=5, n_jobs=-1, verbose=2)
gs_random.fit(X_train_pca, y_train)


# Fitting the Random Forest model with the best parameters
forest2 = RandomForestClassifier(max_depth=20,
max_features='sqrt',
min_samples_leaf=4,
min_samples_split=2, n_estimators=100, random_state=42)
forest2.fit(X_train_pca, y_train)

# Prediction
y_pred_RF = forest2.predict(X_test_pca)

# Evaluate the performance of model
forest_train_accuracy = forest2.score(X_train_pca, y_train)
forest_test_accuracy = forest2.score(X_test_pca, y_test)

print(f'Training data accuracy: {forest_train_accuracy:.2f}')
print(f'Test data accuracy: {forest_test_accuracy:.2f}')

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_RF)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
```

```
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()



#%% SVM

# Creating a dataframe with selected PCs for the training data
pcaDF_train = pd.DataFrame(data=X_train_pca, columns=[f'PC{i+1}' for i in range(n_components)])

# Creating a dataframe with selected PCs for the test data
pcaDF_test = pd.DataFrame(data=X_test_pca, columns=[f'PC{i+1}' for i in range(n_components)])

#Initialize the SVM model with an RBF kernelon PCA transformed trainign data
svmModel = SVC(kernel='rbf', probability=True, random_state=42)
svmModel.fit(pcaDF_train, y_train)

#Evaluation after training has been done
# Make predictions on the test set
y_pred = svmModel.predict(pcaDF_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')

# Print the classification report
print('Classification Report:')
print(classification_report(y_test, y_pred))

# Print the confusion matrix
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))


#%%
# Tuning SVM model
# Defining basic pipeline with StandardScaler, PCA, and SVM to perform grid search
pipeline = Pipeline([
    ('scaler', StandardScaler()),  # Standardization
    ('pca', PCA(n_components=6)),  # PCA with the selected number of components
    ('svm', SVC(kernel='rbf', probability=True, random_state=42))  # SVM
])

# Performing 5-fold cross-validation on the entire dataset
cv_scores = cross_val_score(pipeline, X, y, cv=5, scoring='accuracy')

# Output the individual fold scores
print(f'Cross-Validation Scores: {cv_scores}')

# Output the mean accuracy and standard deviation
print(f'Mean Cross-Validation Accuracy: {cv_scores.mean():.4f}')
print(f'Standard Deviation of Cross-Validation Accuracy: {cv_scores.std():.4f}')
```

```
# Define the pipeline with StandardScaler, PCA, and SVM
pipeline = Pipeline([
    ('scaler', StandardScaler()),   # Standardization
    ('pca', PCA(n_components=6)),
    ('svm', SVC(probability=True))  # SVM
])

# Expanding the parameter grid for more comprehensive tuning
paramGrid = {
    'svm__C': [0.1, 1, 10, 100, 1000],  # Regularization parameter
    'svm__gamma': ['scale', 'auto', 0.1, 0.01, 0.001, 0.0001],  # Kernel coefficient
    'svm__kernel': ['rbf', 'poly', 'sigmoid']  # Different kernel types
}

# Set up GridSearchCV
grid_search = GridSearchCV(estimator=pipeline, param_grid=paramGrid,
cv=5, scoring='accuracy', verbose=2, n_jobs=-1)

# Fit the GridSearch to the data
grid_search.fit(X_train_pca, y_train)

# Print the best parameters and the best accuracy score
print(f'Best Parameters: {grid_search.best_params_}')
print(f'Best Cross-Validation Accuracy: {grid_search.best_score_:.4f}')

# Evaluate the best model on the test set
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_pca)

# Calculate accuracy
svm_accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy on Test Set: {accuracy:.4f}')

# Print the classification report and confusion matrix
print('Classification Report:')
print(classification_report(y_test, y_pred))



#%% Logistic Regression

# Initialize and train the Logistic Regression model
log_reg = LogisticRegression()
log_reg.fit(X_train_pca, y_train)

# Predict the test set results
y_pred = log_reg.predict(X_test_pca)
accuracy1 = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy1:.2f}")


#%% Tuning Logistic regression model
# Defining sigmoid function
def logistic_loss(X, y, beta, lambda_reg):
    # Logistic loss with L2 regularization
    predictions = expit(X @ beta)
    loss = -np.mean(y * np.log(predictions + 1e-15) + (1 - y) * np.log(1 - predictions + 1e-15))
    regularization = (lambda_reg / 2) * np.sum(beta[1:] ** 2)  #
    Regularize only non-intercept terms
    return loss + regularization

def compute_gradients(X, y, beta, lambda_reg):
    # Compute gradients of the logistic loss
    predictions = expit(X @ beta)
    errors = predictions - y
    gradient = X.T @ errors / len(y)
    gradient[1:] += lambda_reg * beta[1:]  # Add gradient of regularization term
    return gradient

def compute_hessian(X, y, beta, lambda_reg):
    # Compute the Hessian of the logistic loss
    predictions = expit(X @ beta)
    diag = predictions * (1 - predictions)
    H = X.T @ (diag[:, np.newaxis] * X) / len(y)
    H[1:, 1:] += lambda_reg * np.eye(H.shape[0] - 1)  # Add Hessian of regularization term
    return H

def newton_raphson(X, y, lambda_reg, max_iter=100, tol=1e-6):
    beta = np.zeros(X.shape[1])  # Initialize beta
    for _ in range(max_iter):
        gradient = compute_gradients(X, y, beta, lambda_reg)
        H = compute_hessian(X, y, beta, lambda_reg)
        delta = np.linalg.solve(H, gradient)
        beta -= delta
        if np.linalg.norm(delta) < tol:
            break
    return beta

# Prepare data for optimization
X_train_pca_with_intercept = np.hstack([np.ones((X_train_pca.shape[0], 1)), X_train_pca])
X_test_pca_with_intercept = np.hstack([np.ones((X_test_pca.shape[0], 1)), X_test_pca])

lambda_reg = 1.0

# Perform Newton-Raphson optimization
beta_optimized = newton_raphson(X_train_pca_with_intercept, y_train, lambda_reg)

# Predict using the optimized beta
def predict(X, beta):
    # Adding intercept term for prediction
    X_with_intercept = np.hstack([np.ones((X.shape[0], 1)), X])
    return expit(X_with_intercept @ beta) >= 0.5

# Evaluate the model
y_pred_newton = predict(X_test_pca, beta_optimized)
accuracy_newton = accuracy_score(y_test, y_pred_newton)
print(f"Newton-Raphson Optimization Accuracy: {accuracy_newton:.2f}")


#%% Logistic Regression with Elastic Net regularization
```

```
elastic_log_reg = LogisticRegression(penalty='elasticnet',
solver='saga', l1_ratio=0.5, C=1.0, max_iter=10000)
elastic_log_reg.fit(X_train_pca, y_train)

# Predict the test set results
y_pred_elastic = elastic_log_reg.predict(X_test_pca)
accuracy_elastic = accuracy_score(y_test, y_pred_elastic)
print(f"Accuracy with Elastic Net: {accuracy_elastic:.2f}")



#%% kNN

knn = KNeighborsClassifier(n_neighbors=35, p=1, metric='minkowski')
knn.fit(X_train_pca, y_train)
t_pred = knn.predict(X_test_pca)

knn_accuracy_test = knn.score(X_test_pca, y_test)
knn_accuracy_train = knn.score(X_train_pca, y_train)
print(f"Accuracy score for Test-set: {knn_accuracy_test}")
print(f"Accuracy score for Train-set: {knn_accuracy_train}")


#%%
# Finding the most optimal parameteres for number of neighbours
accuracies_test_pca = list()
accuracies_train_pca = list()
accuracies_test = list()
accuracies_train = list()

k = list()
for i in range(50):
    knn = KNeighborsClassifier(n_neighbors=i + 1, p=1, metric='minkowski')
    knn.fit(X_train_pca, y_train)
    t_pred_pca = knn.predict(X_test_pca)
    accuracy_test_pca = knn.score(X_test_pca, y_test)
    accuracy_train_pca = knn.score(X_train_pca, y_train)

    knn.fit(X_train, y_train)
    t_pred = knn.predict(X_test)
    accuracy_test = knn.score(X_test, y_test)
    accuracy_train = knn.score(X_train, y_train)

    accuracies_test_pca.append(accuracy_test_pca)
    accuracies_train_pca.append(accuracy_train_pca)
    accuracies_test.append(accuracy_test)
    accuracies_train.append(accuracy_train)
    k.append(i + 1)

plt.figure(figsize=(13, 5))
plt.plot(k, accuracies_test_pca, "-", label="Test-set, after PCA")
plt.plot(k, accuracies_train_pca, "-", label="Train-set, after PCA")
plt.plot(k, accuracies_test, "-", label="Test-set, without PCA")
plt.plot(k, accuracies_train, "-", label="Train-set, without PCA")
plt.ylabel("Accuracies")
plt.xlabel("k")
plt.legend()
plt.grid(1)

print(
    f"Highest accuracy of Test-set without PCA:
    {max(accuracies_test)}, with k value of:
    {k[accuracies_test.index(max(accuracies_test))]}")
print(
    f"Highest accuracy of Test-set with PCA:
    {max(accuracies_test_pca)}, with k value of :
    {k[accuracies_test_pca.index(max(accuracies_test_pca))]}")


#%% Tuning kNN for the most optimal scores possible

test_sizes = np.arange(0.1, 0.6, 0.05)
k_values = np.arange(1, 40, 1)

# Plotting possible accuracy results
accuracy_results = np.zeros([len(test_sizes), len(k_values)])
for i, test_size in enumerate(test_sizes):
    for j, k in enumerate(k_values):
        X_train, X_test, t_train, t_test = train_test_split(X, y, test_size=test_size,
        random_state=42)
        knn = KNeighborsClassifier(n_neighbors=k, p=1, metric='minkowski')
        knn.fit(X_train, t_train)
        accuracy = np.round(knn.score(X_test, t_test), 3)
        accuracy_results[i, j] = accuracy

plt.figure(figsize=(20,5))
sns.heatmap(accuracy_results, annot=True, fmt=".2f", xticklabels=k_values,
yticklabels=np.round(test_sizes,2), cmap='viridis', annot_kws={"size":8})
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Test Dataset Size')
plt.title('Grid Search: Test Dataset Size vs k-Value')
plt.show()

#%% Final optimized kNN model

knn2 = KNeighborsClassifier(n_neighbors=35, p=1, metric='minkowski')
knn2.fit(X_train_pca, y_train)
t_pred = knn2.predict(X_test_pca)

accuracy_test = knn2.score(X_test_pca, y_test)
accuracy_train = knn2.score(X_train_pca, y_train)

print(f"Accuracy score for Test-set: {np.round(accuracy_test, 3)}")
print(f"Accuracy score for Train-set: {np.round(accuracy_train,3)}")

precision = precision_score(y_test, t_pred, average='weighted')
recall = recall_score(y_test, t_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Precision: {np.round(precision,3)}")
print(f"Recall: {np.round(recall,3)}")
print(f"F1 Score: {np.round(f1,3)}")
print("Classification Report:\n", classification_report(y_test, t_pred))


conf_matrix = confusion_matrix(y_test, t_pred)
plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=set(y_test),
yticklabels=set(y_test))
plt.title('Confusion Matrix')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.show()



#%% Comparison

# The modell names and corresponding accuracy values
model_names = ["K-Nearest Neighbors", "Logistic Regression", "Random Forest", "SVM (RBF Kernel)"]
accuracies = [knn_accuracy_test, accuracy_elastic, forest_test_accuracy, svm_accuracy]

performance_df = pd.DataFrame({
    "Model": model_names,
    "Test Accuracy": accuracies
})

# Printing table showing accuracy scores
print(performance_df)

# Plotting the accuracies of the models
plt.figure(figsize=(10, 6))
plt.barh(model_names, accuracies, color=['navy', 'mediumblue', 'royalblue', 'cornflowerblue'])
plt.xlabel('Test Accuracy')
plt.title('Model Comparison Based on Test Accuracy')
plt.xlim(0, 1)  # Assuming accuracy is between 0 and 1
plt.grid(False)
plt.tight_layout()
plt.show()
```