Dyson School of Design Engineering

Data Science Report

Predicting the Presence of Heart Disease based on Health Conditions

Xinyue, Zhang	Yujeong, Seo	Zhihan, Wang	Minseo (Erik), Kim
02217044	02272316	02243294	02284802

Table of Contents

Abstract

This report performs predictive analysis on the presence of heart disease using various health parameters. The dataset contains the comprehensive health information, such as chest pain type, resting blood pressure, serum cholesterol, thalassemia type.

The aim of this report is to develop multiple predictive models using the dataset, to accurately predict heart disease. Four methods include: logistic regression, decision tree, random forest and support vector machine. While all four methods showed promising results, the decision tree method appeared to have the best accuracy and recall. The result of the analysis presents the noticeable relationship between the presence of heart disease and following variables: the number of major vessels coloured by fluoroscopy, thalassemia disease and the maximum heart rate.

1. Introduction

Cardiovascular diseases (CVDs), a general term for heart diseases, are a leading global cause of mortality, with an estimated 17.9 million deaths each year. [1] With its prevalence continuing to rise, the number of deaths has increased to 18.6 million, comprising one-third of total annual deaths. [2] While the heart attack or stroke may be the first sign of diseases, 80% of CVD deaths are due to heart attacks and strokes. Thus heart disease can be sudden, and so early detection via existing health conditions can provide potential patients with treatments to prevent premature deaths.

2. Background

A number of scientific researches have been conducted to predict the contribution of risk factors. Previous study on the same dataset [3] compares the different machine learning techniques. Four methods, Naïve Bayes' classifier, decision tree, K-nearest neighbour and random forest have been analysed for its accuracy of predicting heart disease. The Naïve Bayes and K-nearest neighbour with k-value of 7 provided the highest accuracy of 88.1% and 90.8%, respectively. This research can be expanded by introducing and comparing new methods, as well as comparing the precision and recall, considering the problem context.

3. Methodology

3.1 Data Selection

UCI Heart Disease Data from kaggle is chosen for the analysis. [4] [5] The dataset contains 16 attributes and a total of 920 samples. The attributes contain basic age and sex, to health conditions such as resting blood pressure, maximum heart rate, and more. The patient's heart disease is represented in a category of 0 to 4.

3.2 Data Pre-processing

3.2.1 Filtering the data

The dataset is reduced to only the Cleveland database with 304 samples due to missing columns in other locations. Further filtering the incomplete sample leaves 297 data samples.

3.2.2 Eliminating the attributes

In dataset preparation for analysis, irrelevant attribute 'id' was eliminated. Also, since all filtered 297 samples are located in Cleveland, the 'origin' is deemed irrelevant and removed.

Also, the attributes 'oldpeak' (exercise induced ST reduction) and 'slope' (the slope of ST segment), both have connections to ST, however combining these would obscure their individual influences to heart disease. Hence these were kept separated.

3.2.3 Binarising the data

In our dataset, heart disease levels were categorised into five distinct classes: 0, 1, 2, 3, and 4. For the purposes of the prediction, the scale is binarised: presence (1) and no presence (0). Not only the dataset contained significantly few instances of levels 2, 3, and 4. This simplification allows for a more straightforward and meaningful analysis.

4. Analysis and Discussion

4.1 Logistic Regression - Erik

4.1.1 Aim

To develop a model that can predict whether a patient has heart disease based on their independent features.

4.1.2 Methodology

Logistic regression was chosen to find the association of dependant variables to the binary dependant variable. In this case, whether a patient has heart disease or not is classified as either positive or negative; a binary data. Of the two methods for features selection, the logistic regression forward selection was preferred due its ability to build a model with high sensitivity that identifies correlating variables to heart disease. Backward elimination was not chosen because it was challenging to determine which features had a weak correlation with heart disease due to the limitation of professional medical knowledge.

4.1.2.1 Filtering the Data

Before processing the data, the data was analysed to check for dependent variables or features to be eliminated to ensure the validity of the model's assumptions. Oldpeak and slope seemed related since both were related to the ST segment on an ECG. However, oldpeak referred to the amount of depression, while Slope referred to the pattern of change. [7] Also, all missing data and insignificant attributes were eliminated in 3.2.2. Hence, no additional filtering was conducted.

4.1.2.2 Balancing the Data

A confusion matrix was generated for the dataset to assess whether there was imbalance in the data. No imbalances were found with 109 positive cases and 144 negative cases. With a total sample number of 298, The proportion of positives is approximately 36.6% and the proportion of negatives is approximately 48.3%. Therefore, undersampling was unnecessary as reducing the sample size could lead to the inaccuracy of the prediction.

4.1.2.3 Splitting the Data

The data was split into a training set and a validation / test set, with a ratio of 60% to 40% as per the average ratio used for logistic regression. The validation / test set was divided into halves; 20:20. The accuracy of the final model is influenced by how the data is divided, so additional split ratios were tested, including an 80/20 split and a 70/30 split.

4.1.3 Results

4.1.3.1 Feature selection

For application of the automatic forward selection algorithm as per chosen in 4.1.1, a model with no variables was created. Through an iterative process, features were sequentially assessed based on their correlation with "num". Each feature identified with the strongest correlation was added to the model. If the inclusion of a feature resulted in a validation accuracy improvement of 0.5%, it was retained; otherwise, it was discarded. This process continued until no further features contributed to the model's performance.

4.1.3.2 Validation

5 models were selected through forward selection, and their performance was evaluated by applying them to the test set to assess their accuracy.

The three figures below illustrate the association between model accuracy and the number of features added to the model. *Fig 1 (60/40)* depicts slight overfitting until 3 features, when the validation accuracy surpasses the training accuracy. This can happen if there is a high dropout in the model or if the split ratio is not optimal. [8] In such cases, since morel features are used in testing compared to the 60% used in training, validation accuracy may be higher. *Fig 2 (70/30)* depicts an excellent fit of both training and validation accuracy, with only a slight overfitting after 10 features were added. *Fig 3 (80/20)* shows an overfit, with a large difference between accuracies. Hence data from logistic regression forward selection of the 70/30 split was carried on to create a model.

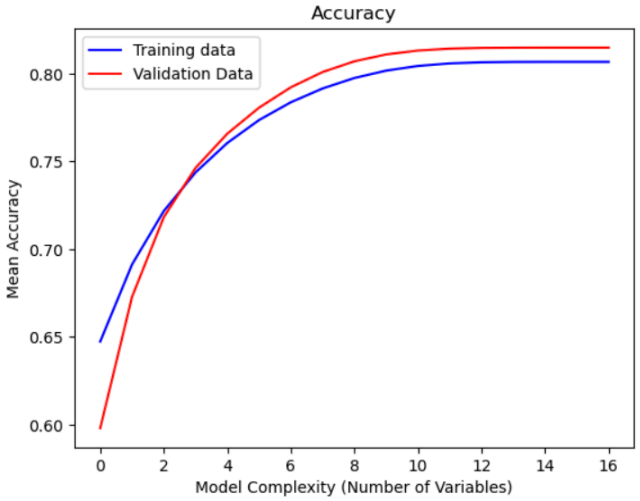

Fig. 1. 60/40 split Model Accuracy vs Number of variants

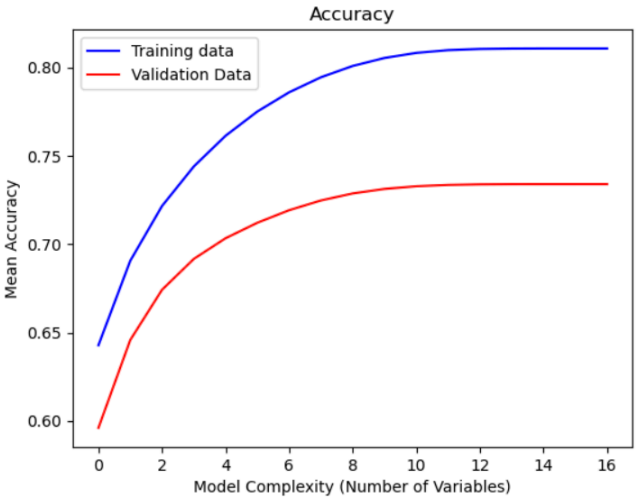

Fig. 2. 70/30 split Model Accuracy vs Number of variants

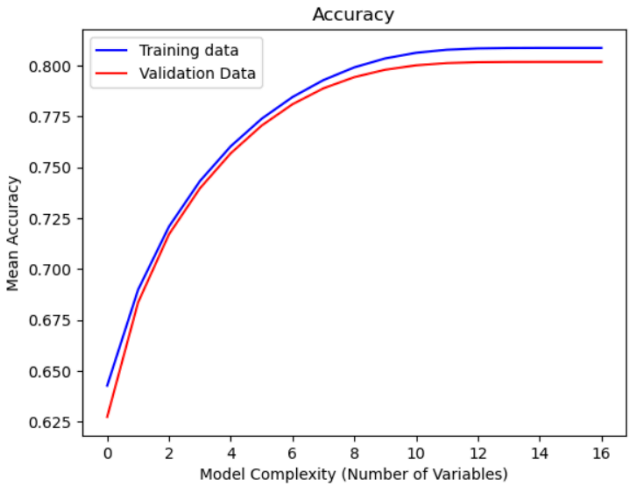

Fig. 3. 80/20 split Model Accuracy vs Number of variants

4.1.4 Discussion

4.1.4.1 Final Results

The factors that had the most influence in whether or not a patient has heart disease was found through logistic regression forward selection. These, in order of relevance, were cp non-anginal, ca, thal reversable defect, exang, cp typical angina, thal normal, restecg normal, slope flat, fbs, oldpeak, cp atypical angina, slope upsloping, age, trestbps, and thalch.

The addition of these features coincided with a decrease in validation accuracy, dropping steadily from an initial 0.78 to a final 0.21. From *table 1 and fig 2*, it can be deduced that the addition of the 7th feature would change the accuracy of the model drastically, which is reinforced by its drop from 0.78 to 0.47.

Table 1: results of Logistic Regression Forward Selection Model

	Accuracy	**Precision**	**Recall**
7 Train	0.889	0.947	0.818
7 Validation	0.756	0.867	0.591
8 Train	0.822	0.889	0.727
8 Validation	0.662	0.6	0.682

4.1.4.2 Precision and Recall

For medical practices, data with high recall, low precision is preferred to data with low recall, high precision. Especially with regards to a life threatening illness such as a heart disease, it is logical to assume that diagnosing a patient with a disease they do not have (false positive) is better than failing to identify a disease (false negative), which could have serious consequences for the patient. Even at the expense of triggering unnecessary stress, the ability to catch a heart disease and ensure a timely treatment is critical.

4.1.4.3 Limitations

The model's accuracy may be enhanced by incorporating backward selection in addition to forward selection. Forward selection can lead to suppressor effects, where predictors only become significant when another predictor is held constant,

thereby reducing accuracy. [9] Also, the process was computationally expensive as it required fitting multiple models to determine the best predictor to add at each step. This became evident in a 17 feature combination function, with computing time reaching around an hour to complete. An introduction of the lasso binary logistic method would also be beneficial by adding a penalty equal to the absolute value of the magnitude of coefficients that can result in sparse models with fewer coefficients to reduce computation load.

4.2 Decision Tree - Zhihan

4.2.1 Aim

Use a trained decision tree model to predict the presence or absence of heart disease for a patient based on the independent features, and identify the most influential factors affecting the prediction.

4.2.2 Methodology

A decision tree is a supervised learning algorithm used for classification problems, in this case predicting the presence or absence of heart disease. It creates a tree-like model with nodes representing features, branches representing decisions, and leaf nodes representing outcomes. In this task, the decision tree was trained using features like age, serum cholesterol and chest pain type. The algorithm identifies the most important features by reducing Gini impurity, which measures the likelihood of incorrectly classifying a data point. The primary aim of the decision tree is to identify the feature that reduces Gini impurity the most, so that pure splits are made and the overall classification accuracy of the tree is improved.

In my analysis, "no heart disease" was set as 0 and "heart disease" as 1. In heart disease prediction, recall is more important than accuracy because it measures the proportion of actual sick people correctly identified by the model. Misclassifying a sick person as healthy (False Negative) can delay treatment and be life-threatening, whereas misclassifying a healthy person as sick (False Positive) is less critical. Therefore, maximising recall ensures that potential heart disease cases are identified, minimising the risk of missing a diagnosis.

4.2.3 Data Preparation

4.2.3.1 Data Balance

A specific consideration was made regarding the 'restecg' feature, which has three unique values: ‘normal’, ‘stt abnormality’, and ‘lv hypertrophy’. In the dataset of 297 data points, only four samples had the ‘stt abnormality’ value. Despite its clinical importance [10], the low frequency suggests it may not significantly contribute to the model . [11] Retaining these rare instances could lead to underfitting. Therefore, the instances with the ‘restecg’ value of ‘stt abnormality’ were eliminated before analysis.

Based on the remaining 293 samples, the class distribution shows that 45.73% are positive (have heart disease) and 54.27% are negative (do not have heart disease). The difference in class proportions is relatively small, with only an 8.54% difference, indicating a fairly balanced dataset.

4.2.3.2 Feature Engineering

In the group data processing section, the dataset was cleaned by eliminating data with missing features and binarization of the target variable. All binary features were transformed into 0 and 1. Nominal data, including ‘cp’, ‘thal’ and ‘slope’ were transformed into a numerical format using one-hot encoding, making them suitable for decision tree. [12] ‘cp_1’, ‘thal_1’ and ‘slope_1’ were randomly removed to avoid multicollinearity, so that high correlation between the encoded variables would be eliminated. All Training data was standardised.

4.2.3.3 Data Split

To prevent overfitting, the dataset was randomly split into three parts: 80% for training, 10% for validation, and 10% for testing. The training set was used to build the decision tree model, the validation set helped fine-tune the model and the testing set was used to evaluate the performance.

4.2.4 Maximum Depth

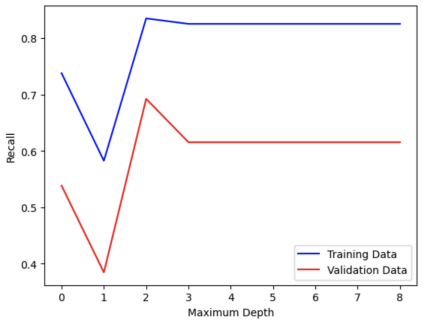

Fig. 4 . Maximum Depth vs Recall

Hyperparameters are user-defined settings that shape the structure of decision trees and help prevent overfitting. To find the optimal values for maximum depth and minimum impurity decrease, the training and validation sets are used, evaluating their impact on recall by plotting the results. As the complexity of the model increased, the recall on both the training and validation sets fluctuated following the same trend. This indicates that the model is not overfitting and is generalising well to new data. Therefore, we chose a maximum depth of 2, as it achieved high recall on both the training and validation sets.

4.2.5 Minimum Impurity Decrease

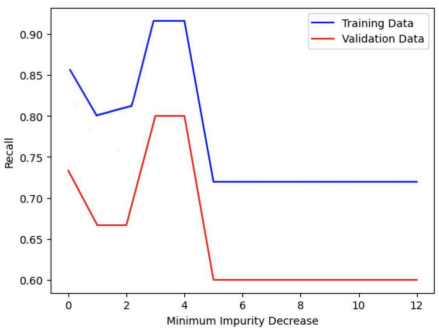

Fig. 5 . Minimum Impurity Decrease vs Recall

Fig. 5 demonstrates that the model does not significantly overfit the training data. The difference between the training and validation sets remains relatively constant. Therefore, the Minimum Impurity Decrease was set to 0, which ensures a higher recall level while maintaining good generalisation to new data.

4.2.6 Results

The decision tree illustrated in Fig. 6 was constructed using the identified hyperparameters. The decision tree illustrated in Fig. 6 was constructed using the identified hyperparameters. The Gini coefficients of the leaf nodes ranged from 0.038 to 0.499. The model showed a test recall of 88.9% and a training recall of 86.4%.

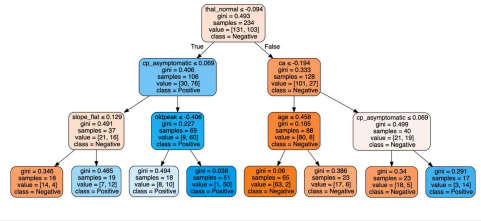

Fig. 6. Decision Tree

Table 2. *Model Performance Metric data*

	Accuracy	Precision	Recall
Training	0.846	0.819	0.864
Validation	0.759	0.75	0.733
Testing	0.933	1	0.889

Table 2 illustrates that the model's accuracy and recall on the test set exceeded those on the training set, as well as the majority of true positives and true negatives shown in the confusion matrix. This makes the model a reliable predictor. Notably, 73.3% of actual sick people were correctly identified by the model, demonstrating its effectiveness in generalising to independent datasets.

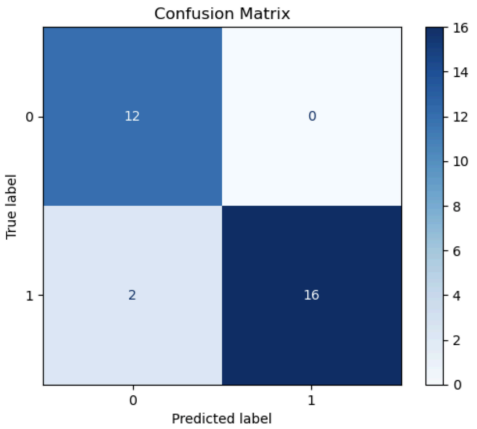

Fig. 7. Confusion Matrix

4.2.7 Discussion

The varying Gini values in the decision tree illustrate the model's confidence in different scenarios. The lowest Gini value of 0.038 means that patients with chest pain symptoms and the slope of the peak exercise ST segment are not flat and with high oldpeak always have heart disease. This high confidence contributes to a high number of true positives (TP). Whereas higher Gini values indicate uncertainty, leading to mixed outcomes.

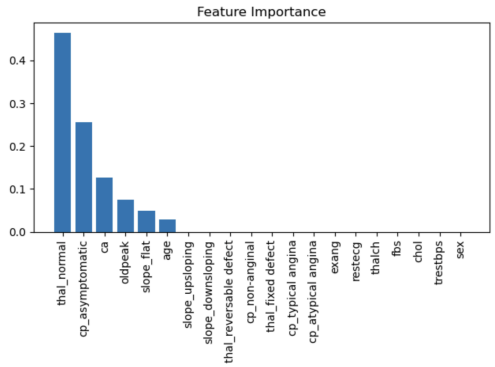

Fig. 6 . Feature Importance

To avoid problems with dummy variables and multicollinearity, one category from each nominal feature was randomly removed during the initial one-hot encoding process. This reduced the 20 features to 17. This might have excluded important features, which would affect the model's performance and the importance. A separate plot of feature importance was made, from which the most significant features are the status of thalassemia, the chest pain type, the slope of the peak exercise ST segment, the oldpeak value, the number of major vessels, and the patient's age.

The thalassemia status emerged as the most important feature, indicating a strong correlation that can assist medical professionals in identifying whether patients have heart diseases or not.

In conclusion, the model performed well on the test set, but there are some limitations. Four samples with "stt abnormality" were removed because they were rare, which could affect the model's performance on such cases. The gender distribution in the dataset was not very balanced, which might affect the model's effectiveness for male and female patients. While the results are promising overall, further steps should be taken to address these limitations. This could involve collecting more balanced and comprehensive data. This will help doctors to more accurately predict the presence of heart disease.

4.3 Random Forest - Yujeong

4.3.1 Aim

This section predicts the effect of different health condition parameters on the presence of heart disease using random forest method.

4.3.2 Methodology

Random forest is a supervised machine learning model that consists of multiple decision trees. The results of decision trees from randomly chosen rows and columns are accumulated and competed to reach a single result. Its randomness, known as bagging, results in low correlation between trees, thus reducing the potential overfitting risks. With multiple attributes considered, random forest is useful for evaluating and comparing the contribution of each variable. [13]

Recall values are compared as the increase in recall represents the reduction in false negatives. Only after the parameters are optimised for higher recall values, the final model will go through the testing set.

4.3.2.1 Filtering the data

The presence of heart disease is binarised as 0 for negative and 1 for positive, and any attributes with missing data have been removed in group processing.

As the random forest feature in scikit-learn requires the input values to be numerical, qualitative attributes are converted into quantitative scales or are removed. The slope of the peak exercise ST segment (slope) attributes are removed as it can not be classified into distinctive or continuous states. Chest pain type (cp), resting electrocardiographic results (restecg) and blood disorder thalassemia (thal) attributes are binarised, separating the normal, 0 and abnormal, 1, i.e. the presence of any symptoms in the category.

4.3.2.2 Balancing and Splitting the data

Checking the balance, the dataset contains 137 positives and 160 negatives for the target variable. The ratio of smaller class over larger class equals 0.86, thus, the dataset is balanced and the negative class does not have to be undersampled.

The dataset is randomly split into training, validating, and testing sets in a ratio of 60:20:20.

4.3.3 Random Forest Model

A random forest classifier is constructed to tune three hyperparameters: maximum depth, minimum samples split, and the number of estimators.

An initial model starts from the parameter values of max_depth = 2, min_samples_split = 0.1 and random_state = 0. One hyperparameter is tested at a time, with each iteration introducing new constraints. Each model aims to maximise the recall value.

4.3.3.1 Hyperparameter Tuning - Maximum depth

This parameter represents the maximum depth, the depth from the root to the leaf, of the tree.

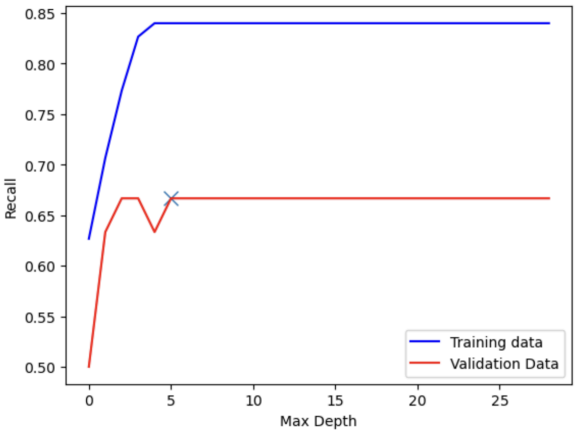

Fig 7. Maximum Depth vs Recall

While the deeper tree can be trained by greater amounts of data, a higher chance of overfitting exists. On the other hand, lower value makes the tree simple due to insufficient data. Thus, despite the validation recall reaching its maximum earlier when the max_depth is 3, max_depth = 5 is chosen where the value starts converging.

4.3.3.2 Hyperparameter Tuning - Minimum Sample Split

Minimum sample split represents the lowest number of samples required to split an internal node.

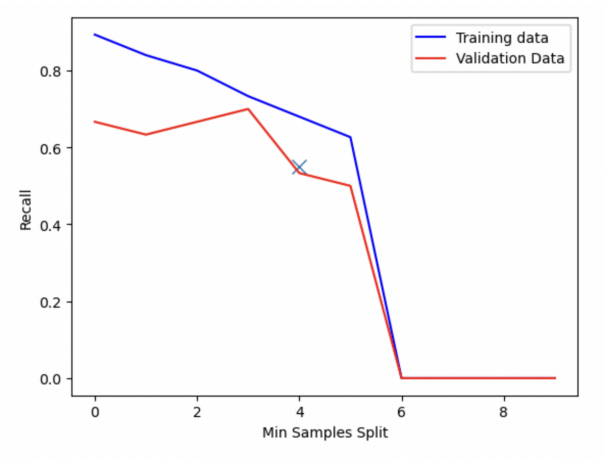

Fig. 8. Minimum Samples Split vs Recall

Minimum sample split prevents trees creating the branches from very few samples. Thus, the smaller the minimum sample split, the more likely to create a biased model, tailored to training data. Reversely, value beyond 6 shows a significant drop in recall due to simplified models. To ensure the performance on validation and testing, a minimum sample split value of 4 is carried forward.

4.3.3.3 Hyperparameter Tuning - N-Estimators

N-Estimator determines the number of decision trees in the forest.

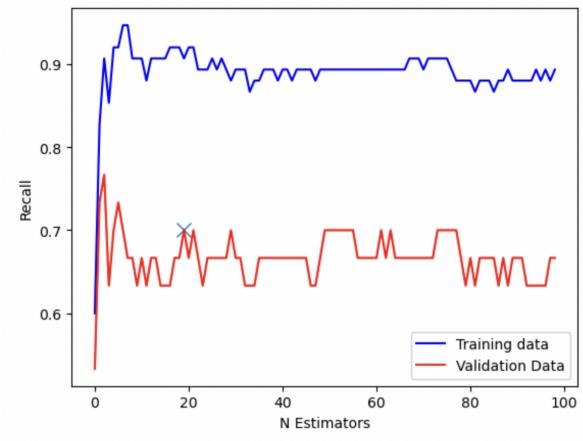

Fig. 9. N-Estimators vs Recall

The small number of n-estimators may fail to capture the complexity of the model, while a larger number is likely to enhance the performance but accompanies computational time and costs. Thus n-estimators value of 19 is chosen as a balance between two ends.

4.3.4 Results

After the tuning, the final model has the following parameters: max_depth = 5, min_samples_split = 4, n_estimators = 19, random_state = 0.

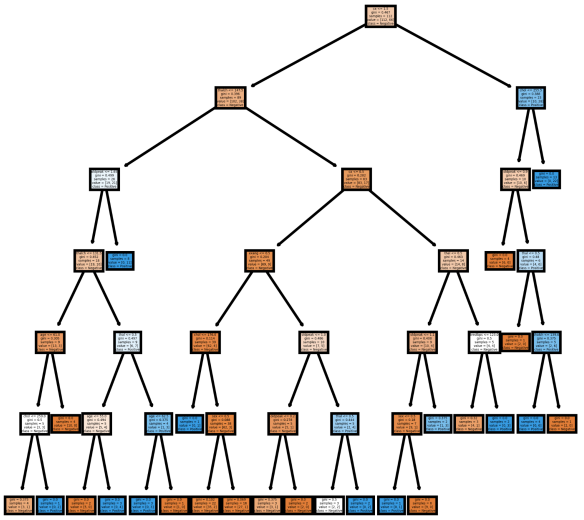

Fig. 10. Decision Tree

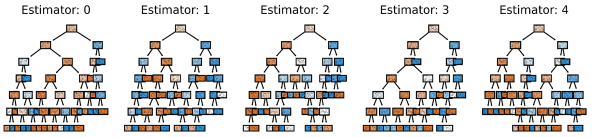

Fig. 11. Random Forest

Table 3. Comparison of Recall between the Models

Model	Training	Validation	Testing
1	0.707	0.633	0.687
2	0.84	0.633	0.719
3	0.893	0.667	0.656
4	0.92	0.667	0.781

Models evolved by adding one new hyperparameter in each iteration. Model 4, the final model, achieved the highest recall scores: 0.92 for training, 0.667 for validation, and 0.781 for testing. High performance on the training set alone might indicate overfitting but the final model demonstrates a balanced performance across all three sets, thus, model 4 is carried forward.

4.3.5 Final Results

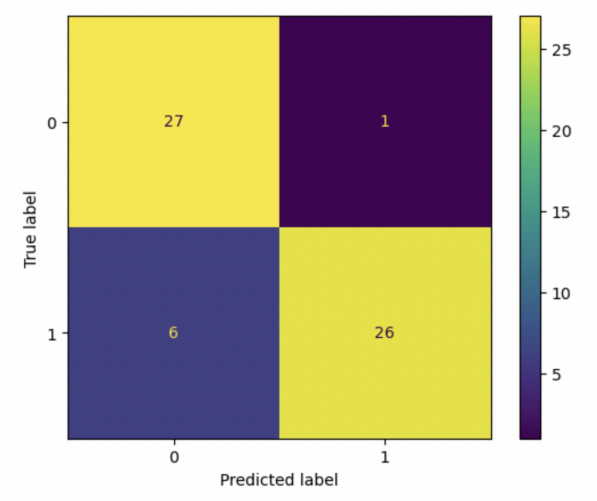

Fig. 12. Confusion Matrix

Confusion matrix of the final model is plotted to assist the understanding on the performance of the random forest. Indicated by the colour label of the matrix, the number of false negatives and false positives is low, consisting of 11.6% of the testing set, thus reducing the chance of false diagnosis.

Table 4. Results of Final Model

	Accuracy	**Precision**	**Recall**
Training	0.938	0.932	0.92
Validation	0.779	0.869	0.667
Testing	0.867	0.962	0.781

On the validation set, the model maintains the solid performance on accuracy, precision and recall, suggesting reliable generalisation despite some opportunity for improvement. This small discrepancy might be caused due to the limitation of a small data set. The improved performance on the testing set proves its efficiency on unseen data.

The high precision, especially notable for training and testing sets, highlights the model's ability to minimise false positives, as well as reasonable ability to reduce false negatives supported by the recall values.

4.3.6 Discussion

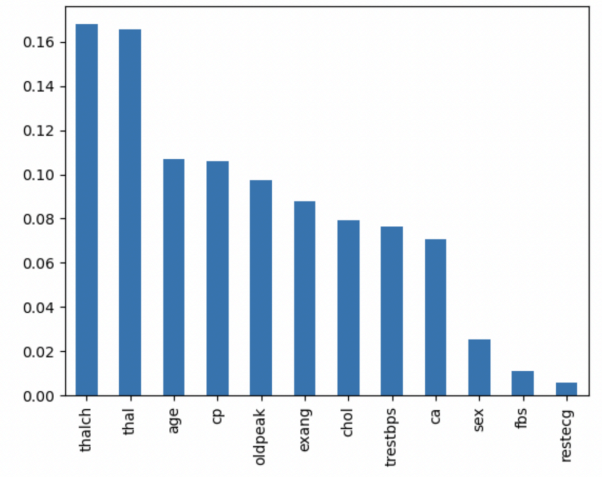

Fig. 13. Feature Importance

The feature importance plot suggests that the most significant predictors of heart disease are: maximum heart rate (thalch), the blood disorder thalassemia (thal) and age. While the significance of age is similar to that of chest pain (cp) and following variables, the first two features stand out among the features. The result underscores that compared to other conditions, cardiovascular health conditions are directly associated with the likelihood of having heart disease.

The final model exhibits a reasonably high precision of 0.962 and a recall of 0.781, enabling accurate predictions. However, due to the small sample size, it may struggle when applied on larger datasets. Training with a larger dataset could mitigate potential performance issues as well as the risk of overfitting.

Furthermore, employing the GridSearchCV method allows for more comprehensive hyperparameter tuning. Whereas the tuning is done manually on section 4.3.3, GridSearchCV automatically evaluates all possible combinations to determine the optimal hyperparameters. [14]

4.4 Support Vector Machine - Xinyue (Alina) Zhang

4.4.1 Aim

This section aims to develop a prediction model that assesses patients' heart disease risks by comprehensively analysing clinically measured data and patient context. This is vital in early intervention considering that most heart diseases does not show obvious symptoms until heart attack occurs.[15]

4.4.2 Methodology

The Support Vector Machine (SVM) model is chosen for its outstanding capability to handle complex classification variables, making it optimal for analysing multiple features of heart disease risk. [15] SVMs aim to find the optimal hyperplane that separates classes with the maximum margin, enhancing the model's generalizability to new data and reducing the risk of over-fitting. A kernelized SVM is used to handle nonlinear inputs and analyse the relationships between features.

4.4.3 Data Preparation

4.4.3.1 Feature Engineering

During pre-processing, missing data was efficiently removed, ensuring all features are usable. However, Sklearn SVM cannot handle non-numerical data, therefore all binary data is transformed to '0' and '1'. Additionally, categorical data such as chest pain type (cp), resting electrocardiographic results (restecg), the slope of the peak exercise ST segment (slope), and blood disorder thalassemia (thal) were binarized.

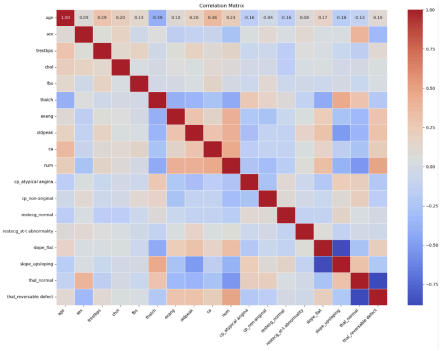

Fig. 14. Correlation Matrix

By analysing the correlation matrix, multicollinearity was identified between 'thal-normal' and 'thal-reversible defects', and between 'slope_upsloping' and 'slope_flat', leading to arbitrary removal. Categories both representing abnormalities were combined. Irrelevant features, including resting blood pressure (trestbps), serum cholesterol (chal), and fasting blood sugar (fbs), were removed to reduce noise and over-fitting.

4.4.3.2 Data Balancing

A confusion matrix revealed a minor imbalance in the dataset, with 137 positive cases and 160 negative cases. Due to the limited number of samples, the dataset was kept to ensure sufficient training and validation.

4.4.3.2 Data Splitting

The data is standardised, shuffled, and split into a 70% training set, a 15% validation set, and a 15% testing set after trying different combinations. The training set is used to determine the parameters of the model. The validation set is used to compare with the training results to evaluate the chosen parameters. Finally, the testing set is used to assess the overall performance of the final model.

4.4.4 Hyperparameter selection

Different values of three hyperparameters—kernel type, C-value, and gamma value—are tested and optimised for the model. In the context of heart disease prediction, healthcare providers typically prioritise minimising false negatives, even if this results in more false positives. Missing a diagnosis can have fatal consequences, so early detection is crucial for effective treatment, even if it occasionally results in a false diagnosis. [16] Therefore, the selection logic focuses on maximising recall to minimise false negatives, while also optimising precision and accuracy to acceptable levels.

4.4.4.1 Kernel Type Selection

Different kernel types transform input data into higher-dimensional spaces in distinct ways, making it easier to separate between classes. This enhances the model's ability to handle various data patterns and complexities.[17] Four kernel functions—linear,

polynomial, radial basis function (RBF), and sigmoid—are tested with default values of C = 1 and gamma = 'scale' to determine which is most compatible with the dataset.

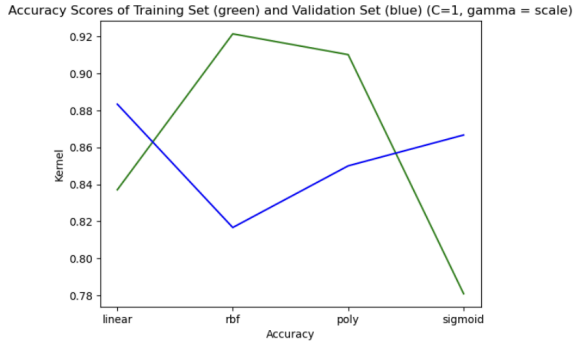

Fig. 15. Accuracy Scores of Training Set & Validation Set (varying kernel)

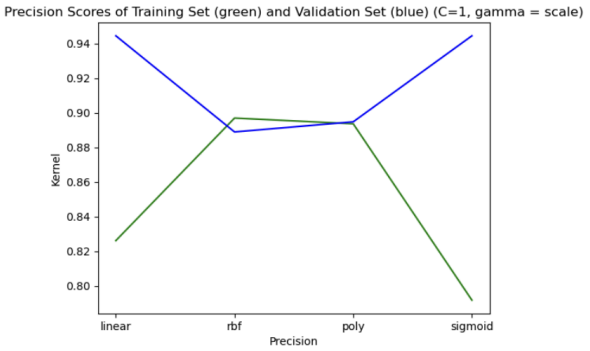

Fig. 15. Precision Scores of Training Set & Validation Set (varying kernel)

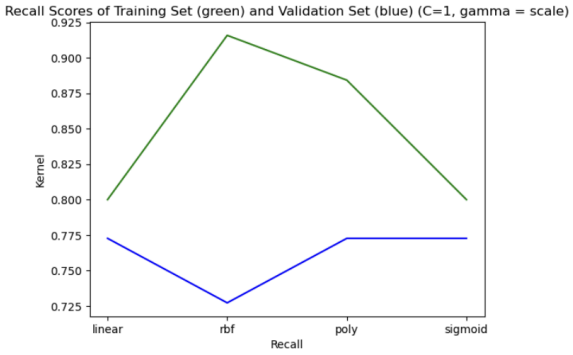

Fig. 16. Recall Scores of Training Set & Validation Set (varying kernel)

Sigmoid and linear kernels that show unknown fit are eliminated for selection. Polynomial kernel performs the best, achieving prediction accuracy of 85%, precision of 89%, and recall of 0.775.

4.4.4.2 C Value Selection

C value is a regularisation parameter that controls the trade-off between achieving a low error on the training data and minimising the norm of the weights, thereby performing significant impact on the fitting.[18] C values ranging from 0 to 10 are tested and selected comparing trade-offs between the precision and recall values.

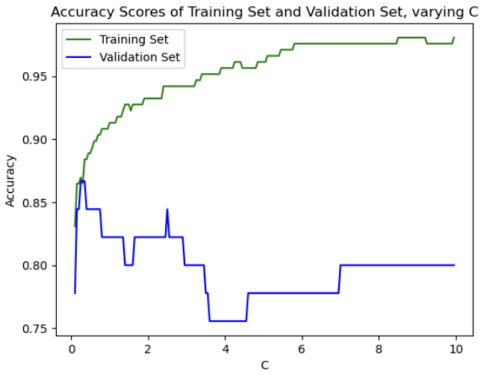

Fig. 18. Accuracy Scores of Training Set & Validation Set (varying C)

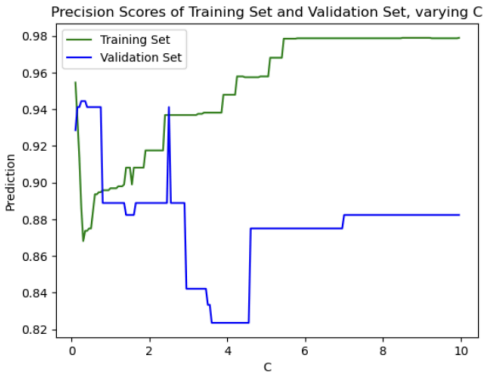

Fig. 19. Precision Scores of Training Set & Validation Set (varying C)

Fig. 20. Recall Scores of Training Set & Validation Set (varying C)

The fluctuation with small C value reflects the noise in the data, indicating underfitting. Conversely, selecting a high C value might lead to overfitting due to increased sensitivity to noise during training (shown as the decrease with high C value).[19] Therefore, C = 2 is selected to balance between underfitting and overfitting, with recall = 0.723.

4.4.4.3 Gamma Value Selection

Similarly, gamma values are selected from 0.01 to 0.1.

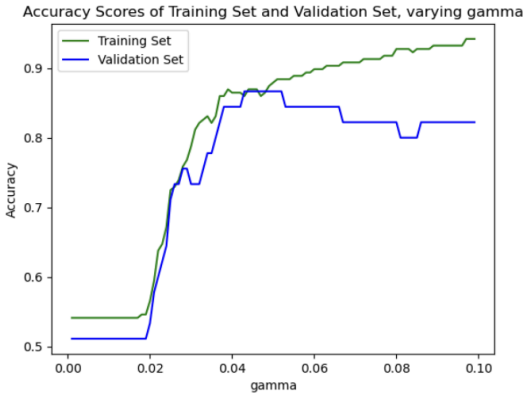

Fig. 21. Accuracy Scores of Training Set & Validation Set (varying gamma)

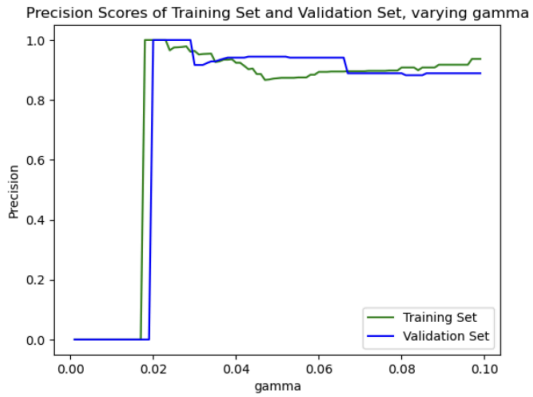

Fig. 22. Precision Scores of Training Set & Validation Set (varying gamma)

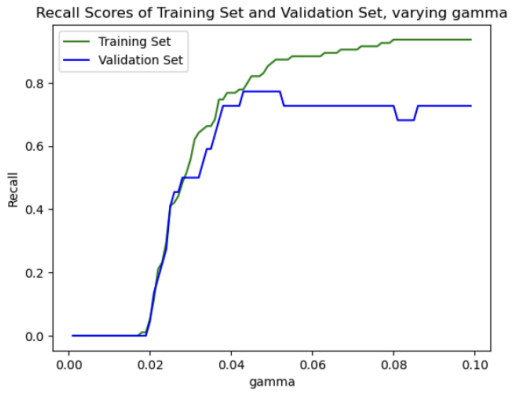

Fig. 23. Recall Scores of Training Set & Validation Set (varying gamma)

The gamma value reflects the radius of influence of the support vectors.[18] Higher gamma values could result in overfitting of data, therefore gamma = 0.05 is selected, which increases the prediction accuracy to 86.7% and recall to 0.772.

4.4.5 Final Results

The test result of the final model is below:

Table 5. Model Performance Metrics

	Training	Validation	Testing
Accuracy	0.879	0.867	0.733
Precision	0.872	0.944	0.722
Recall	0.863	0.772	0.65

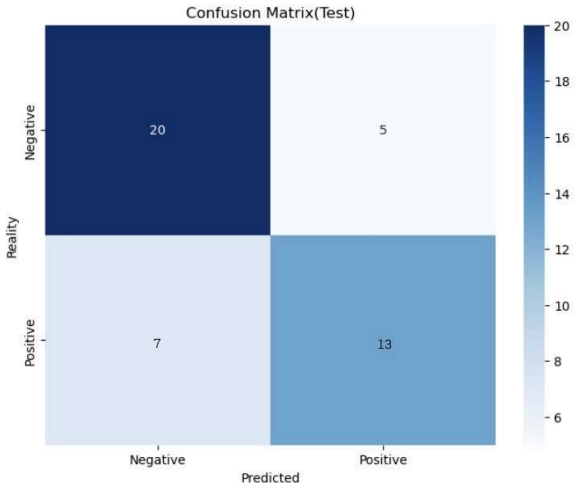

Fig. 24. Confusion matrix for testing data

4.4.6 Discussion

4.4.6.1 The final results

Overall, the polynomial SVM's performance is mediocre in predicting heart disease from the dataset, achieving an accuracy of 73.3% and a precision of 72.2%. Compared to the training dataset, the recall score of the testing set is significantly lower, with the recall value = 0.65, introducing a significant number of false negatives. Including the F1-score and AUC can help balance between the precision and recall. The deviance could also be the consequence of insufficient dataset, a larger dataset can be used to increase the effectiveness of the model and reduce bias when splitting the dataset.

4.4.6.2 PCA ranking

By conducting a Principal Component Analysis (PCA), the importance of each feature is ranked. The top five features are the slope of the peak exercise ST segment (slope), maximum heart rate (thalch), age,

exercise-induced angina (exang), and sex. Future development of the model could focus on these parameters to reduce noise and prevent overfitting from irrelevant features and to reduce dimensionality. [20] This approach is also beneficial for reducing the time and effort spent measuring insignificant parameters.

4.4.6.3 Limitations

It is important to note that the patient data used in this study comes solely from the V.A. Medical Center, Long Beach, and the Cleveland Clinic Foundation, after filtering out incomplete records. Consequently, the dataset may not be applicable to other regions, as lifestyle factors such as diet and physical inactivity significantly contribute to the high prevalence of heart disease in the U.S.[21] For more generalised predictions, future studies should include a more diverse geographical dataset.

5. Conclusion

Considering that each analysis requires distinct further processing and serves different purposes, it is not conclusive that one method outperforms others. However, comparing metrics can suggest which model was more successful in predictions.

Recall is prioritised as the false negatives, missed diagnoses, costs more than false positives in life-threatening circumstances. The decision tree method has the highest accuracy (0.933) and recall (0.889). Accuracy and recall of the models can be enhanced by training with larger samples. Also, the uniform background of patients limits the generalisation of the results to the global population.

By analysing the correlation heat map, the most positively correlated features with heart disease are the number of major vessels coloured by fluoroscopy (ca) and the most negatively correlated features are normal thalassemia (thal normal) and maximum heart rate (thalch). It corresponds to the results of analysis: ‘ca’ from logistic regression and decision tree, and ‘thal’ and ‘thalch’ from decision tree, random forest, and SVM.

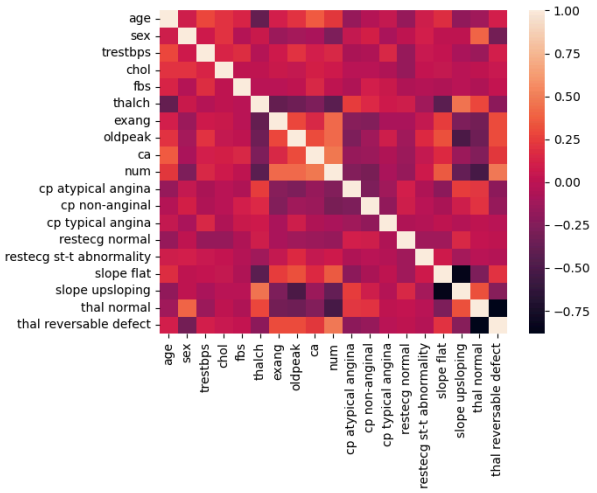

Fig. 25. Correlation Heatmap

6. References

Group

[1] World Health Organization, “Cardiovascular diseases (CVDs),” World Health Organization, Jun. 11, 2021. https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds)

[2] M. Peng et al., “Prediction of cardiovascular disease risk based on major contributing features,” Scientific Reports, vol. 13, no. 1, p. 4778, Mar. 2023, doi: https://doi.org/10.1038/s41598-023-31870-8.

[3] D. Shah, S. Patel, and S. K. Bharti, “Heart Disease Prediction using Machine Learning Techniques,” SN Computer Science, vol. 1, no. 6, Oct. 2020, doi: https://doi.org/10.1007/s42979-020-00365-y.

[4] “UCI Heart Disease Data,” www.kaggle.com. https://www.kaggle.com/datasets/redwankarimsony/heart-disease-data/data

[5] “UCI Machine Learning Repository,” archive.ics.uci.edu. https://archive.ics.uci.edu/dataset/45/heart+disease

[6] M. C. Gerson, S. N. Morris, and P. L. McHenry, “Relation of exercise-induced physiologic S-T segment depression to R wave amplitude in normal subjects,” The American Journal of Cardiology, vol. 46, no. 5, pp. 778–782, Nov. 1980, doi: https://doi.org/10.1016/0002-9149(80)90428-2.

Logistic Regression - Erik

[7] M. S. Amin, Y. K. Chiam, and K. D. Varathan, “Identification of significant features and data mining techniques in predicting heart disease,” ELSEVIER, vol. 36, p. 82~93, Mar. 2019.

[8] “Why is my validation accuracy higher than training accuracy? | Kaggle,” Kaggle.com, 2024. https://www.kaggle.com/discussions/questions-and-answers/250648 (accessed Jun. 24, 2024).

[9] G. Borboudakis, “Forward-Backward Selection with Early Dropping,” Journal of Machine Learning Research, vol. 20, pp. 1–39, 2019, Available: https://jmlr.org/papers/volume20/17-334/17-334.pdf

Decision Tree - Zhihan

[10] T. Ohira et al., “Prospective Study of Major and Minor ST-T Abnormalities and Risk of Stroke Among Japanese,” Stroke, vol. 34, no. 12, Dec. 2003, doi: https://doi.org/10.1161/01.str.0000103742.83117.fb.

[11] W. Lian, G. Nie, B. Jia, D. Shi, Q. Fan, and Y. Liang, “An Intrusion Detection Method Based on Decision Tree-Recursive Feature Elimination in Ensemble Learning,” Mathematical Problems in Engineering, vol. 2020, pp. 1–15, Nov. 2020, doi: https://doi.org/10.1155/2020/2835023.

[12] M. Saberian, P. Delgado, and Y. Raimond, “Gradient Boosted Decision Tree Neural Network,” arXiv.org, Nov. 05, 2019. https://arxiv.org/abs/1910.09340

Random Forest - Yujeong

[13] “What is Random Forest? | IBM,” www.ibm.com. https://www.ibm.com/topics/random-forest?mhsrc=ibmsearch_a&mhq=random%20forest

[14] K. Sandunil, Z. Bennour, H. Ben Mahmud, and A. Giwelli, “Effects of Tuning Hyperparameters in Random Forest Regression on Reservoir’s Porosity Prediction. Case Study: Volve Oil Field, North Sea,” onepetro.org, Jun. 25, 2023. https://onepetro.org/ARMAUSRMS/proceedings/ARMA23/All-ARMA23/ARMA-2023-0660/532467 (accessed Jun. 24, 2024).

SVM - Xinyue (Alina)

[15] Sakinat Oluwabukonla Folorunso, Joseph Bamidele Awotunde, Emmanuel Abidemi Adeniyi, Kazeem Moses Abiodun, and Femi Emmanuel Ayo, “Heart Disease Classification Using Machine Learning Models,” Communications in computer and information science, pp. 35–49, Jan. 2022, doi: https://doi.org/10.1007/978-3-030-95630-1_3.

[16] D. Duprez’, “Early detection of cardiovascular disease - the future of cardiology?,” www.escardio.org. https://www.escardio.org/Journals/E-Journal-of-Cardiology-Practice/Volume-4/vol4n19-Title-Early-detection-of-cardiovascular-disease-the-future-of-cardi

[17] B. Dhiyanesh, S. Ganapathi Ammal, K. Saranya, and K. E. Narayana, “Advanced Cloud-Based Prediction Models for Cardiovascular Disease: Integrating Machine Learning and Feature Selection Techniques,” SN Computer Science/SN computer science, vol. 5, no. 5, May 2024, doi: https://doi.org/10.1007/s42979-024-02927-w.

[18] Scikit-learn, “Scaling the regularization parameter for SVCs,” scikit-learn. https://scikit-learn.org/stable/auto_examples/svm/plot_svm_scale_c.html (accessed Jun. 24, 2024).

[19] S. B. Kotsiantis, “Supervised Machine Learning: A Review of Classification Techniques,” Informatica, vol. 31, no. 3, 2007, Available: https://www.informatica.si/index.php/informatica/article/view/148

[20] R. Cross, Principal component analysis. Clanrye Intl, 2015.

[21] University of Texas Medical Branch at Galveston, “Heart attack mortality rate higher in the US compared to other high-income countries,” ScienceDaily, May 05, 2022. https://www.sciencedaily.com/releases/2022/05/220505143829.htm

7. Appendix (codes)

7.1 Group Pre-Processing

```
import pandas as pd
heart = pd.read_csv('heart_disease_uci.csv')

# removing the samples
heart.drop(heart[heart['origin'] != 'Cleveland'].index, inplace=True)
heart.dropna(inplace=True)

# removing attributes
del heart['id']
del heart['origin']

# binarising non-numerical values
heart['sex'] = heart['sex'].map({'male': 0, 'female': 1})
heart['fbs'] = heart['fbs'].map({FALSE: 0, TRUE: 1})
heart['exang'] = heart['exang'].map({FALSE: 0, TRUE: 1})

# binarising the level of heart disease to its presence
heart['num'] = heart['num'].map(lambda x: 1 if x != 0 else 0)
```

7.2 Logistic Regression - Erik

7.2.1 Data setup

```
# Import Libraries
from sklearn.linear_model import LogisticRegression as logreg
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score
import pandas as pd
import matplotlib.pyplot as plt
import warnings
import itertools

# Surpress warnings
warnings.filterwarnings("ignore")

# Read csv
heart = pd.read_csv('heart.csv')
```

7.2.2 Additional Feature engineering

```
# categorical variable into a binary

# cp
heart['cp atypical angina'] = heart['cp'] == 'atypical angina'
```

```
heart['cp atypical angina'] = heart['cp atypical angina'].astype(int)

heart['cp non-anginal'] = heart['cp'] == 'non-anginal'
heart['cp non-anginal'] = heart['cp non-anginal'].astype(int)

heart['cp typical angina'] = heart['cp'] == 'typical angina'
heart['cp typical angina'] = heart['cp typical angina'].astype(int)

# restecg
heart['restecg normal'] = heart['restecg'] == 'normal'
heart['restecg normal'] = heart['restecg normal'].astype(int)

heart['restecg st-t abnormality'] = heart['restecg'] == 'st-t abnormality'
heart['restecg st-t abnormality'] = heart['restecg st-t abnormality'].astype(int)

# slope
heart['slope flat'] = heart['slope'] == 'flat'
heart['slope flat'] = heart['slope flat'].astype(int)

heart['slope upsloping'] = heart['slope'] == 'upsloping'
heart['slope upsloping'] = heart['slope upsloping'].astype(int)

# thal
heart['thal normal'] = heart['thal'] == 'normal'
heart['thal normal'] = heart['thal normal'].astype(int)

heart['thal reversable defect'] = heart['thal'] == 'reversable defect'
heart['thal reversable defect'] = heart['thal reversable defect'].astype(int)
```

7.2.3 ModelSummary Class

```
# ModelSummary Class provided
from scipy import stats
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, accuracy_score

class ModelSummary:
    """ This class extracts a summary of the model

    Methods
    -------
    get_se()
        computes standard error
    get_ci(SE_est)
        computes confidence intervals
    get_pvals()
```

```
    computes p-values
get_summary(name=None)
    prints the summary of the model
"""

def __init__(self, clf, X, y):
    """
    Parameters
    ----------
    clf: class
        the classifier object model
    X: pandas Dataframe
        matrix of predictors
    y: numpy array
        matrix of variable
    """
    self.clf = clf
    self.X = X
    self.y = y
    pass

def get_se(self)
    predProbs = self.clf.predict_proba(self.X)
    X_design = np.hstack([np.ones((self.X.shape[0], 1)), self.X])
    V = np.diagflat(np.product(predProbs, axis=1))
    covLogit = np.linalg.inv(np.dot(np.dot(X_design.T, V), X_design))
    return np.sqrt(np.diag(covLogit))

def get_ci(self, SE_est):
    """
    Parameters
    ----------
    SE_est: numpy array
        matrix of standard error estimations
    """
    p = 0.975
    df = len(self.X) - 2
    crit_t_value = stats.t.ppf(p, df)
    coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
    upper = coefs + (crit_t_value * SE_est)
    lower = coefs - (crit_t_value * SE_est)
    cis = np.zeros((len(coefs), 2))
    cis[:,0] = lower
    cis[:,1] = upper
    return cis

def get_pvals(self):
    p = self.clf.predict_proba(self.X)
```

```
    n = len(p)
    m = len(self.clf.coef_[0]) + 1
    coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
    se = self.get_se()
    t =  coefs/se
    p = (1 - stats.norm.cdf(abs(t))) * 2
    return p

def get_summary(self, names=None):
    ses = self.get_se()
    cis = self.get_ci(ses)
    lower = cis[:, 0]
    upper = cis[:, 1]
    pvals = self.get_pvals()
    coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
    data = []
    for i in range(len(coefs)):
        currlist = []
        currlist.append(np.round(coefs[i], 3))
        currlist.append(np.round(ses[i], 3))
        currlist.append(np.round(pvals[i], 3))
        currlist.append(np.round(lower[i], 3))
        currlist.append(np.round(upper[i], 3))
        data.append(currlist)
    cols = ['coefficient', 'std', 'p-value', '[0.025', '0.975]']
    sumdf = pd.DataFrame(columns=cols, data=data)
    if names is not None:
        new_names = ['intercept']*(len(names) + 1)
        new_names[1:] = [i for i in names]
        sumdf.index = new_names
    else:
        try:
            names = list(self.X.columns)
            new_names = ['intercept']*(len(names) + 1)
            new_names[1:] = [i for i in names]
            sumdf.index = new_names
        except:
            pass
    print(sumdf)
    acc = accuracy_score(self.y, self.clf.predict(self.X))
    confmat = confusion_matrix(self.y, self.clf.predict(self.X))
    print('-'*60)
    print('Confusion Matrix (total:{}) \t Accuracy: \t  {}'.format(len(self.X),np.round(acc, 3)))
    print('  TP: {} | FN: {}'.format(confmat[1][1],confmat[1][0]))
    print('  FP: {} | TN: {}'.format(confmat[0][1],confmat[0][0]))
```

7.2.4 Confusion Matrix

```
# Confusion matrix
X = heart[['age', 'cp atypical angina', 'cp non-anginal', 'cp typical angina', 'trestbps', 'chol', 'fbs', 'restecg normal', 'restecg st-t abnormality', 'thalch', 'exang', 'oldpeak', 'slope flat', 'slope upsloping', 'ca', 'thal normal', 'thal reversable defect']]
y = heart[['num']]

mylr = logreg()
mylr.fit(X, y)

mylrsummary = ModelSummary(mylr, X, y)
mylrsummary.get_summary()
```

7.2.5 Splitting the Data

```
# Put 70% of the data into a training set and 30% into a combined testing set
X_train, X_test_val, y_train, y_test_val = train_test_split(X, y, test_size=0.3, shuffle = True, random_state = 0)

# Split the combined testing set into validation and testing sets
X_val, X_test, y_val, y_test = train_test_split(X_test_val, y_test_val, test_size=0.5, random_state= 0)
```

7.2.6 Logistic Regression

```
# Logistic regression
def select_column_to_add(X_train, y_train,X_val, y_val, columns_in_model, columns_to_test):
    column_best = None
    columns_in_model = list(columns_in_model)

    if len(columns_in_model)==0:
        acc_best = 0

    elif len(columns_in_model) == 1:
        mylr = logreg().fit(X_train[columns_in_model].values.reshape(-1,1),y_train)
        acc_best = accuracy_score(y_val,mylr.predict(X_val[columns_in_model].values.reshape(-1,1)))

    else:
        mylr = logreg().fit(X_train[columns_in_model], y_train)
        acc_best = accuracy_score(y_val, mylr.predict(X_val[columns_in_model]))

    for column in columns_to_test:
        mylr = logreg().fit(X_train[columns_in_model+[column]],y_train)
        y_pred = mylr.predict(X_val[columns_in_model+[column]])
        acc = accuracy_score(y_val,y_pred)

        if acc - acc_best >= 0.005:
            acc_best = acc
            column_best = column
```

```
    if column_best is not None:
        print('Adding {} to the model'.format(column_best))
        print('The new best validation accuracy is {}'.format(acc_best))
        columns_in_model_updated = columns_in_model + [column_best]

    else:
        print('Did not add anything to the model')
        columns_in_model_updated = columns_in_model

    return(columns_in_model_updated, acc_best)
```

7.2.7 Logistic Regression Forwards Selection

```
# Logistic regression forward selection
def auto_forward_selection(X_train,y_train,X_val,y_val,max_num_of_features):
    columns_to_test = list(X_train.columns)
    columns_in_model = []
    current_acc = 0

    for i in range (0,max_num_of_features):
            columns_in_model, acc_best = select_column_to_add(X_train,y_train,X_val,y_val,columns_in_model,
columns_to_test)

        if acc_best == current_acc:
            break

        else:
            for feature in columns_to_test:
                if feature in columns_in_model:
                    columns_to_test.remove(feature)
                    print(columns_in_model, acc_best)
                return (columns_in_model, acc_best)
```

7.2.8 Running Iterations to Train Model

```
# Running the function
columns_in_model_1, acc_best_1 = auto_forward_selection(X_train, y_train,X_val, y_val,5)

#again ignore feature selected
columns_in_model_2, acc_best_2 = auto_forward_selection(X_train.drop(columns=['ca']),
y_train,X_val.drop(columns=['ca']), y_val,5)

#again
columns_in_model_3, acc_best_3 = auto_forward_selection(X_train.drop(columns=['ca',
'thal reversable defect']), y_train,X_val.drop(columns=['ca', 'thal reversable defect']), y_val,5)

#again
```

```
columns_in_model_4, acc_best_4 = auto_forward_selection(X_train.drop(columns=['ca', 'thal reversable defect', 'thal normal']), y_train,X_val.drop(columns=['ca', 'thal reversable defect', 'thal normal']), y_val,5)

#again
columns_in_model_5, acc_best_4 = auto_forward_selection(X_train.drop(columns=['ca', 'thal reversable defect', 'thal normal', 'restecg normal']), y_train,X_val.drop(columns=['ca', 'thal reversable defect', 'thal normal', 'restecg normal']), y_val,5)
```

7.2.9 Validating Accuracies

```
# Validation of accuracies
means = X_train.mean(axis=0)
stds = X_train.std(axis=0)

X_test_standardised = (X_test - means)/stds
X_train_standardised = (X_train - means)/stds

X1_train = X_train_standardised[columns_in_model_1]
X2_train = X_train_standardised[columns_in_model_2]
X3_train = X_train_standardised[columns_in_model_3]
X4_train = X_train_standardised[columns_in_model_4]
X5_train = X_train_standardised[columns_in_model_5]

mylr1 = logreg()
mylr1.fit(X1_train, y_train)

mylr2 = logreg()
mylr2.fit(X2_train, y_train)

mylr3 = logreg()
mylr3.fit(X3_train, y_train)

mylr4 = logreg()
mylr4.fit(X4_train, y_train)

mylr5 = logreg()
mylr5.fit(X5_train, y_train)

X1_test = X_test_standardised[columns_in_model_1]
X2_test = X_test_standardised[columns_in_model_2]
X3_test = X_test_standardised[columns_in_model_3]
X4_test = X_test_standardised[columns_in_model_4]
X5_test = X_test_standardised[columns_in_model_5]

predicted_y1 = mylr1.predict(X1_test)
predicted_y2 = mylr2.predict(X2_test)
predicted_y3 = mylr3.predict(X3_test)
predicted_y4 = mylr4.predict(X4_test)
```

```
predicted_y5 = mylr5.predict(X5_test)

con_mat1 = confusion_matrix(y_test,predicted_y1, labels=[1,0])
print(con_mat1)

con_mat2 = confusion_matrix(y_test,predicted_y2, labels=[1,0])
print(con_mat2)

con_mat3 = confusion_matrix(y_test,predicted_y3, labels=[1,0])
print(con_mat3)

con_mat4 = confusion_matrix(y_test,predicted_y4, labels=[1,0])
print(con_mat4)

con_mat5 = confusion_matrix(y_test,predicted_y5, labels=[1,0])
print(con_mat5)

print('model 1: ')
print(accuracy_score(y_test,predicted_y1))
print(precision_score(y_test,predicted_y1))
print(recall_score(y_test,predicted_y1))

print('model 2: ')
print(accuracy_score(y_test,predicted_y2))
print(precision_score(y_test,predicted_y2))
print(recall_score(y_test,predicted_y2))

print('model 3: ')
print(accuracy_score(y_test,predicted_y3))
print(precision_score(y_test,predicted_y3))
print(recall_score(y_test,predicted_y3))

print('model 4: ')
print(accuracy_score(y_test,predicted_y4))
print(precision_score(y_test,predicted_y4))
print(recall_score(y_test,predicted_y4))

print('model 5: ')
print(accuracy_score(y_test,predicted_y5))
print(precision_score(y_test,predicted_y5))
print(recall_score(y_test,predicted_y5))

total = len(heart)
hd_pos = heart['num'].sum()
hd_neg = total - hd_pos
```

7.2.10 Plotting Graphs

```
# Plotting graphs
def combinations(attributes,group_size):
    comb = list(itertools.combinations(attributes, group_size))
    return comb

plot_data = {'Acc_train':[],'Prec_train':[], 'Rec_train':[],'Acc_val':[], 'Prec_val':[], 'Rec_val': []}
columns_to_test =['age', 'cp atypical angina', 'cp non-anginal', 'cp typical angina', 'trestbps', 'chol', 'fbs', 'restecg normal', 'restecg st-t abnormality', 'thalch','exang', 'oldpeak', 'slope flat', 'slope upsloping', 'ca', 'thal normal', 'thal reversable defect']

def get_plot_data(column_to_test):

    acc_list = []
    prec_list = []
    rec_list = []
    acc_val_list = []
    prec_val_list = []
    rec_val_list = []

    for i in range(len(column_to_test)):
        trial_predictors = combinations(column_to_test, i+1)

        for col_comb in trial_predictors:
            X = X_train[list(col_comb)]
            X_Test = X_test[list(col_comb)]
            y = y_train
            mylr = logreg()
            mylr.fit(X,y)

            predicted_y = mylr.predict(X)
            acc = accuracy_score(y, predicted_y)
            prec = precision_score(y, predicted_y)
            rec = recall_score(y, predicted_y)

            predicted_y_test = mylr.predict(X_Test)
            acc_val = accuracy_score(y_test, predicted_y_test)
            prec_val = precision_score(y_test, predicted_y_test)
            rec_val = recall_score(y_test, predicted_y_test)

            acc_list.append(acc)
            prec_list.append(prec)
            rec_list.append(rec)
            acc_val_list.append(acc_val)
            prec_val_list.append(prec_val)
            rec_val_list.append(rec_val)
```

```
        plot_data['Acc_train'].append(np.mean(acc_list))
        plot_data['Prec_train'].append(np.mean(prec_list))
        plot_data['Rec_train'].append(np.mean(rec_list))
        plot_data['Acc_val'].append(np.mean(acc_val_list))
        plot_data['Prec_val'].append(np.mean(prec_val_list))
        plot_data['Rec_val'].append(np.mean(rec_val_list))

get_plot_data(columns_to_test)

df = pd.DataFrame(plot_data)
print(df)

plt.figure()
plt.plot(df['Acc_train'], 'b')  # Blue line for training accuracy
plt.plot(df['Acc_val'], 'r')    # Red line for validation accuracy
plt.title('Accuracy')
plt.xlabel('Model Complexity (Number of Variables)')
plt.ylabel('Mean Accuracy')
plt.legend(['Training data', 'Validation Data'])

plt.show()
```

7.3 Decision Tree - Zhihan

7.3.1 Data Setup

```
import pandas as pd

# Read the dataset
bc = pd.read_csv('heart_disease_ucifiltered.csv')

# Replace boolean and gender values with numeric equivalents
bc = bc.replace({
    True: 1,
    False: 0,
    'Male': 1,
    'Female': 0,
})

# Replace 'restecg' categorical values with numeric equivalents
bc['restecg'] = bc['restecg'].replace({
    'normal': 0,
    'lv hypertrophy': 1
})

# Print the first few rows of the dataset and the dataset size
print(bc.head())
```

```
print('Dataset size: {} '.format(len(bc)))

# Split the dataset into features and target variable
y = bc['num']
x = bc.drop(columns='num')
X = pd.get_dummies(x, columns=['cp', 'thal', 'slope'], drop_first=True)
for col in X.columns:
    if X[col].dtype == 'bool':
        X[col] = X[col].astype(int)

# Print the generated column names to check the one-hot encoded columns
print(X)

# Print the modified dataset to check the results
print(X.head())

# Calculate and print the number of attributes (excluding the 'num' column)
print('The number of attributes is: {}'.format(len(X.columns)))

# Calculate and print the proportion of positive and negative classes
nb_pos = bc['num'].sum()
nb_neg = len(bc) - bc['num'].sum()
print('Proportion of positive: {:.2%}, negative: {:.2%} '.format(nb_pos / len(bc), nb_neg / len(bc)))
```

7.3.2 Splitting the Dataset

```
from sklearn.model_selection import train_test_split
X_train, X_test_val, y_train, y_test_val = train_test_split(X, y, test_size=0.2, shuffle = True, random_state = 0)

# Use the same function above for the validation set
X_val, X_test, y_val, y_test = train_test_split(X_test_val, y_test_val, test_size=0.5, random_state= 0)

print('X len: {}'.format(len(X)))
print('X_train: {}, y_train: {}'.format(len(X_train), len(y_train)))
print('X_test,y_test: {}'.format(len(X_test), len(y_test)))
print('X_val: {}, y_val: {}'.format(len(X_val), len(y_val)))
```

7.3.3 Standardise the Dataset

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
# Transform the training, validation, and test sets
X_train = scaler.transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
print(X.columns)
print(X_test)
```

7.3.4 Find Optimal Maximum Depth

```
# Find Optimal Maximum Depth
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.metrics import accuracy_score, precision_score, recall_score
rec_train = []
rec_val = []
for i in range(1,10):
    dt = DecisionTreeClassifier(max_depth=i, min_impurity_decrease=0.01, random_state=0)
    dt.fit(X_train, y_train)
    rec_train.append(recall_score(y_train, dt.predict(X_train)))
    rec_val.append(recall_score(y_val, dt.predict(X_val)))
plt.figure()
plt.plot(rec_train,"b")
plt.plot(rec_val, "r")
plt.xlabel('Maximum Depth')
plt.ylabel('Recall')
plt.legend(['Training Data','Validation Data'])
plt.show()
```

7.3.5 Find Optimal Minimum Impurity Decrease

```
#find optimal Minimum Impurity Decrease
rec_train = []
rec_val = []
for i in range(1,14):
    dt = DecisionTreeClassifier(max_depth=2, min_impurity_decrease=i/100)
    dt.fit(X_train, y_train)
    rec_train.append(recall_score(y_train, dt.predict(X_train)))
    rec_val.append(recall_score(y_val, dt.predict(X_val)))
plt.figure()
plt.plot(rec_train,"b")
plt.plot(rec_val, "r")
plt.xlabel('Minimum Impurity Decrease')
plt.ylabel('Recall')
plt.legend(['Training Data','Validation Data'])
plt.show()
```

7.3.6 Print the results

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.metrics import accuracy_score, precision_score, recall_score
clf1 = tree.DecisionTreeClassifier(max_depth=2, min_impurity_decrease=0)
clf1.fit(X_train,y_train)
print(clf1)
yPredTrain = clf1.predict(X_train)
```

```
yPredVal = clf1.predict(X_val)
yPredTest = clf1.predict(X_test)
accTrain = accuracy_score(y_train, yPredTrain)
precTrain = precision_score(y_train, yPredTrain)
recTrain = recall_score(y_train, yPredTrain)
accVal = accuracy_score(y_val, yPredVal)
precVal = precision_score(y_val, yPredVal)
recVal = recall_score(y_val, yPredVal)
accTest = accuracy_score(y_test, yPredTest)
precTest = precision_score(y_test, yPredTest)
recTest = recall_score(y_test, yPredTest)
print('Accuracy on the train set: {}'.format(accTrain))
print('Precision on the train set: {}'.format(precTrain))
print('Recall on the train set: {}'.format(recTrain))
print('Accuracy on the val set: {}'.format(accVal))
print('Precision on the val set: {}'.format(precVal))
print('Recall on the val set: {}'.format(recVal))
print('Accuracy on the testing set: {}'.format(accTest))
print('Precision on the testing set: {}'.format(precTest))
print('Recall on the testing set: {}'.format(recTest))
```

7.3.7 Visualise Decision Tree

```
import sklearn.tree as tree
import graphviz
dot_data = tree.export_graphviz(clf1, out_file=None)
graph = graphviz.Source(dot_data)

predictors = X.columns
dot_data = tree.export_graphviz(clf1, out_file=None, feature_names = predictors, class_names = ('Negative', 'Positive'), filled = True, rounded = True, special_characters = True)
graph = graphviz.Source(dot_data)
graph
```

7.3.8 Visualise Confusion Matrix

```
from sklearn.metrics import ConfusionMatrixDisplay

disp = ConfusionMatrixDisplay.from_estimator(clf1, X_test, y_test, cmap="Blues")
plt.title("Confusion Matrix")
plt.show()
```

7.3.9 Plot the Feature Importance Bar Chart

```
#feature importance
from sklearn.datasets import make_classification
from sklearn.datasets import make_classification
from sklearn.tree import DecisionTreeClassifier
importances = clf1.feature_importances_
```

```
# Sort feature importances in descending order
indices = np.argsort(importances)[::-1]
# Create the plot
plt.title('Feature Importance')
plt.bar(range(len(indices)), importances[indices], align='center')
plt.xticks(range(X_train.shape[1]), X.columns[indices], rotation=90)
plt.tight_layout()
plt.show()
```

7.4 Random Forest - Yujeong

7.4.1 Data Setup

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score

heart = pd.read_csv('heart_disease_uci.csv')

del heart['slope']

# checking the balance of the data
positives = 0
negatives = 0

for i in range(len(heart)):
    if heart.iloc[i,12] == 1:
        positives += 1
    else:
        negatives += 1

print(positives)
print(negatives)

X = heart.drop(columns = 'num')
y = heart['num']

# separating training set (60%) and temporary set (40%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=0)

# separating validation (20%) and testing set (20%)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=0)
```

7.4.2 Hyperparameter Tuning

```
# Changing the Max Depth
rec_train = []
rec_val = []

for i in range(1, 30):
    dt = RandomForestClassifier(max_depth = i, min_samples_split = 0.1, random_state = 0)
    dt.fit(X_train, y_train)

    # predicted value
    ypred_train = model.predict(X_train)
    ypred_val = model.predict(X_val)
    rec_train.append(recall_score(y_train, dt.predict(X_train)))
    rec_val.append(recall_score(y_val, dt.predict(X_val)))

plt.figure()
plt.plot(rec_train,'b')
plt.plot(rec_val, 'r')
plt.xlabel('Max Depth')
plt.ylabel('Recall')
plt.legend(['Training data','Validation Data'])

plt.scatter(5, np.max(rec_val), s=100, marker = 'x', linewidth = 1)

# Changing the Min Samples Split
rec_train = []
rec_val = []

samples = np.linspace(0.01,1,10, endpoint = True)

for i in samples:
    dt = RandomForestClassifier(max_depth = 5, min_samples_split = i, random_state = 0)
    dt.fit(X_train, y_train)

    # predicted value
    ypred_train = model.predict(X_train)
    ypred_val = model.predict(X_val)
    rec_train.append(recall_score(y_train, dt.predict(X_train)))
    rec_val.append(recall_score(y_val, dt.predict(X_val)))

plt.figure()
plt.plot(rec_train,'b')
plt.plot(rec_val, 'r')
plt.xlabel('Min Samples Split')
plt.ylabel('Recall')
plt.legend(['Training data','Validation Data'])
```

```
plt.scatter(4, 0.6, s=100, marker = 'x',linewidth = 1)

# Changing N-Estimators
rec_train = []
rec_val = []

for i in range (1, 100):
    dt = RandomForestClassifier(max_depth = 5, min_samples_split = 4, random_state = 0, n_estimators = i)
    dt.fit(X_train, y_train)

    # predicted value
    ypred_train = model.predict(X_train)
    ypred_val = model.predict(X_val)
    rec_train.append(recall_score(y_train, dt.predict(X_train)))
    rec_val.append(recall_score(y_val, dt.predict(X_val)))

plt.figure()
plt.plot(rec_train,'b')
plt.plot(rec_val, 'r')
plt.xlabel('N Estimators')
plt.ylabel('Recall')
plt.legend(['Training data','Validation Data'])

plt.scatter(19, 0.7, s=100, marker = 'x', linewidth = 1)
```

7.4.3 Gaining Final Results

```
# Testing 4 models
# model = RandomForestClassifier(max_depth = 2, min_samples_split = 0.1, random_state = 0) # model-1
# model = RandomForestClassifier(max_depth = 5, min_samples_split = 0.1, random_state = 0)  # model-2
# model = RandomForestClassifier(max_depth = 5, min_samples_split = 4, random_state = 0) # model-3
model = RandomForestClassifier(max_depth = 5, min_samples_split = 4, random_state = 0, n_estimators = 19)

model = model.fit(X_train, y_train)

ypred_train = model.predict(X_train)
ypred_val = model.predict(X_val)
ypred_test = model.predict(X_test)

# accuracy, precision, and recall
acc_train = accuracy_score(y_train, ypred_train)
pre_train = precision_score(y_train, ypred_train)
rec_train = recall_score(y_train, ypred_train)

acc_val = accuracy_score(y_val, ypred_val)
pre_val = precision_score(y_val, ypred_val)
rec_val = recall_score(y_val, ypred_val)
```

```
acc_test = accuracy_score(y_test, ypred_test)
pre_test = precision_score(y_test, ypred_test)
rec_test = recall_score(y_test, ypred_test)

print("Training", acc_train, pre_train, rec_train)
print("Validation", acc_val, pre_val, rec_val)
print("Test", acc_test, pre_test, rec_test)

# Confusion Matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, ypred_test)
ConfusionMatrixDisplay(confusion_matrix=cm).plot();

# Feature Importances
feature_importances = pd.Series(model.feature_importances_, index =
X_train.columns).sort_values(ascending=False)
feature_importances.plot.bar();

# Printing Trees
from sklearn import tree

# single tree
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=800)
tree.plot_tree(model.estimators_[0],feature_names = X_train.columns,
               class_names = ('Negative', 'Positive'), filled = True)
fig.savefig('decision-tree.png')

# estimators
fig, axes = plt.subplots(nrows = 1,ncols = 5,figsize = (10,2), dpi=900)
for i in range(0, 5):
    tree.plot_tree(model.estimators_[i], feature_names = X_train.columns,
                   class_names = ('Negative', 'Positive'), filled = True, ax = axes[i])
    axes[i].set_title('Estimator: ' + str(i), fontsize = 11)

fig.savefig('random-forest.png')
```

7.5 SVM - Alina

7.5.1 Setting up the data

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import PCA
import warnings
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder
```

7.5.2 Model Summary class

```
# ModelSummary Class
class ModelSummary:
    """
    This class extracts a summary of the model

    Methods
    ------
    get_se()
        Computes standard error
    get_ci(SE_est)
        Computes confidence intervals
    get_pvals()
        Computes p-values
    get_summary(name=None)
        Prints the summary of the model
    """
    def __init__(self, clf, X, y):
        """
        Parameters
        ---------
        clf: class
            The classifier object model
        X: pandas Dataframe
            Matrix of predictors
        y: numpy array
            Matrix of variable
        """
        self.clf = clf
        self.X = X
        self.y = y

    def get_se(self):
        self.clf.predict_proba(self.X)
                                                                                    # From https://stats.stackexchange.com/questions/89484/how-to-compute-the-standard-errors-of-a-logistic-regressions-coefficient
        predProbs = self.clf.predict_proba(self.X)
```

```
        X_design = np.hstack([np.ones((self.X.shape[0], 1)), self.X])
        V = np.diagflat(np.product(predProbs, axis=1))
        covLogit = np.linalg.inv(np.dot(np.dot(X_design.T, V), X_design))
        return np.sqrt(np.diag(covLogit))

    def get_ci(self, SE_est):
        """
        Parameters
        ---------
        SE_est: numpy array
            Matrix of standard error estimations
        """
        p = 0.975
        df = len(self.X) - 2
        crit_t_value = stats.t.ppf(p, df)
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        upper = coefs + (crit_t_value * SE_est)
        lower = coefs - (crit_t_value * SE_est)
        cis = np.zeros((len(coefs), 2))
        cis[:, 0] = lower
        cis[:, 1] = upper
        return cis

    def get_pvals(self):
        self.clf.predict_proba(self.X)
        # From https://stackoverflow.com/questions/25122999/scikit-learn-how-to-check-coefficients-significance
        predProbs = self.clf.predict_proba(self.X)
        n = len(predProbs)
        m = len(self.clf.coef_[0]) + 1
        se = self.get_se()
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        t = coefs / se
        p = (1 - stats.norm.cdf(abs(t))) * 2
        return p

    def get_summary(self, names=None):
        ses = self.get_se()
        cis = self.get_ci(ses)
        lower = cis[:, 0]
        upper = cis[:, 1]
        pvals = self.get_pvals()
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        data = []
        for i in range(len(coefs)):
                currlist = [np.round(coefs[i], 3), np.round(ses[i], 3), np.round(pvals[i], 3), np.round(lower[i], 3),
np.round(upper[i], 3)]
            data.append(currlist)
        cols = ['coefficient', 'std', 'p-value', '[0.025', '0.975]']
```

```
    sumdf = pd.DataFrame(columns=cols, data=data)
    if names is not None:
        new_names = ['intercept'] + names
        sumdf.index = new_names
    else:
        try:
            names = list(self.X.columns)
            new_names = ['intercept'] + names
            sumdf.index = new_names
        except:
            pass
    print(sumdf)
    acc = accuracy_score(self.y, self.clf.predict(self.X))
    confmat = confusion_matrix(self.y, self.clf.predict(self.X))
    print('-' * 60)
    print('Confusion Matrix (total:{}) \t Accuracy: \t  {}'.format(len(self.X), np.round(acc, 3)))
    print('  TP: {} | FN: {}'.format(confmat[1][1], confmat[1][0]))
    print('  FP: {} | TN: {}'.format(confmat[0][1], confmat[0][0]))
```

7.5.2 Importing data

```
# remove warnings
warnings.filterwarnings('ignore')

# reading the datase
data = pd.read_csv("heart_disease.csv")
```

7.5.3 Feature Engineering

7.5.3.1 Binarising dataset

```
# Binarising Dataset
# Identify categorical features
categorical_features = ['cp', 'restecg', 'slope', 'thal']
# Perform one-hot encoding and drop the first category for features with more than 2 categories
data_encoded = pd.get_dummies(data, columns=categorical_features, drop_first=True)
data_encoded = data_encoded.drop('cp_typical angina', axis=1)
#Binarising to 0 and 1
data_encoded = data_encoded.replace({True: 1, False: 0})
```

7.5.3.1 Correlation matrix

```
# Compute the correlation matrix
correlation_matrix = data_encoded.corr()
# Print the correlation matrix
print(correlation_matrix)
```

7.5.3.2 Removing unwanted Feature

```
# Removing features unwanted after feature engineering
data_encoded=data_encoded.drop(['trestbps','chol','fbs','restecg_st-t
abnormality','slope_upsloping','thal_reversable defect'], axis=1)
data_encoded.head()
```

7.5.3.3 Splitting the data

```
# splitting the independent and dependent variables
X = data_encoded.drop('num', axis = 1)
y = data_encoded['num']

# generating confusion matrix
mod = LogisticRegression()
mod.fit(X, y)
modsummary = ModelSummary(mod, X, y)
modsummary.get_summary()
total = len(data_encoded)

#positive and negative dataset
total = len(data_encoded)
pos = data_encoded['num'].sum()
neg = total - pos

# splitting by 70%, 30%
X_train, X_test_val, y_train, y_test_val = train_test_split(X, y, test_size=0.3, shuffle=True, random_state=0)

# splitting by 70%, 15%, 15%
X_test, X_val, y_test, y_val = train_test_split(X_test_val, y_test_val, test_size=0.5, random_state=0)
print('The sizes for train, validation, test should be {}'.format((len(X_train), len(X_val), len(X_test))))
```

7.5.3.4 Standardizing the data

```
# Standardize the datasets
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

7.5.4 Testing different kernel

```
# Default SVM model
model = SVC(probability=True, C=1, kernel='linear', gamma='scale')
model.fit(X_train, y_train)



# Model comparison with different kernels
## accuracy
```

```
accuracytrain = []
accuracyval = []
for k in ('linear', 'poly', 'rbf', 'sigmoid'):
    model = SVC(kernel=k, C = 1, gamma='scale')
    model.fit(X_train, y_train)
    print(k)
    y_predtrain = model.predict(X_train)
    accuracytrain.append(accuracy_score(y_train, y_predtrain))
    y_predval = model.predict(X_val)
    accuracyval.append(accuracy_score(y_val, y_predval))

kernelyaxis = ['linear', 'rbf', 'poly', 'sigmoid']
plt.plot(kernelyaxis, accuracytrain, 'g', label = 'Training')
plt.plot(kernelyaxis, accuracyval, 'b', label = 'Validation')
plt.xlabel('Accuracy')
plt.ylabel('Kernel')
plt.title('Accuracy Scores of Training Set (green) and Validation Set (blue) (C=1, gamma = scale)')


## precision
precisiontrain = []
precisionval = []
for k in ('linear', 'poly', 'rbf', 'sigmoid'):
    model = SVC(kernel=k, C = 1, gamma='scale')
    model.fit(X_train, y_train)
    print(k)
    y_predtrain = model.predict(X_train)
    precisiontrain.append(precision_score(y_train, y_predtrain))
    y_predval = model.predict(X_val)
    precisionval.append(precision_score(y_val, y_predval))

kernelyaxis = ['linear', 'rbf', 'poly', 'sigmoid']
plt.plot(kernelyaxis, precisiontrain, 'g', label = 'Training')
plt.plot(kernelyaxis, precisionval, 'b', label = 'Validation')
plt.xlabel('Precision')
plt.ylabel('Kernel')
plt.title('Precision Scores of Training Set (green) and Validation Set (blue) (C=1, gamma = scale)')

## recall
recalltrain = []
recallval = []
for k in ('linear', 'poly', 'rbf', 'sigmoid'):
    model = SVC(kernel=k, C = 1, gamma='scale')
    model.fit(X_train, y_train)
    print(k)
    y_predtrain = model.predict(X_train)
    recalltrain.append(recall_score(y_train, y_predtrain))
    y_predval = model.predict(X_val)
```

```
    recallval.append(recall_score(y_val, y_predval))

kernelyaxis = ['linear', 'rbf', 'poly', 'sigmoid']
plt.plot(kernelyaxis, recalltrain, 'g', label = 'Training')
plt.plot(kernelyaxis, recallval, 'b', label = 'Validation')
plt.xlabel('Recall')
plt.ylabel('Kernel')
plt.title('Recall Scores of Training Set (green) and Validation Set (blue) (C=1, gamma = scale)')
```

7.5.5 Testing varying C-value

```
# Testing varying C
accuracy_train_c = []
accuracy_val_c = []
precision_train_c = []
precision_val_c = []
recall_train_c = []
recall_val_c = []
y_data_c = []
for c in np.arange(0.1, 10, 0.05):
    y_data_c.append(c)
    model = SVC(kernel='poly', C=c, gamma = 'scale')
    model.fit(X_train, y_train)
    y_predtrain = model.predict(X_train)
    accuracy_train_c.append(accuracy_score(y_train, y_predtrain))
    recall_train_c.append(recall_score(y_train, y_predtrain))
    precision_train_c.append(precision_score(y_train, y_predtrain))
    y_predval = model.predict(X_val)
    accuracy_val_c.append(accuracy_score(y_val, y_predval))
    recall_val_c.append(recall_score(y_val, y_predval))
    precision_val_c.append(precision_score(y_val, y_predval))

plt.plot(y_data_c, accuracy_train_c, 'g', label='Training Set')
plt.plot(y_data_c, accuracy_val_c, 'b', label='Validation Set')
plt.ylabel('Accuracy')
plt.xlabel('C')
plt.title('Accuracy Scores of Training Set and Validation Set, varying C')
plt.legend()
plt.show()

plt.plot(y_data_c, precision_train_c, 'g', label='Training Set')
plt.plot(y_data_c, precision_val_c, 'b', label='Validation Set')
plt.ylabel('Prediction')
plt.xlabel('C')
plt.title('Precision Scores of Training Set and Validation Set, varying C')
plt.legend()
plt.show()
```

```
plt.plot(y_data_c, recall_train_c, 'g', label='Training Set')
plt.plot(y_data_c, recall_val_c, 'b', label='Validation Set')
plt.ylabel('Recall')
plt.xlabel('C')
plt.title('Recall Scores of Training Set and Validation Set, varying C')
plt.legend()
plt.show()
```

7.5.6 Testing varying gamma-value

```
# Training with varying gamma
accuracy_train_g = []
accuracy_val_g = []
precision_train_g = []
precision_val_g = []
recall_train_g = []
recall_val_g = []
y_data_g = []

for g in np.arange(0.001, 0.1, 0.001):
    y_data_g.append(g)
    model = SVC(kernel='poly', C=2, gamma = g)
    model.fit(X_train, y_train)
    y_predtrain = model.predict(X_train)
    accuracy_train_g.append(accuracy_score(y_train, y_predtrain))
    recall_train_g.append(recall_score(y_train, y_predtrain))
    precision_train_g.append(precision_score(y_train, y_predtrain))
    y_predval = model.predict(X_val)
    accuracy_val_g.append(accuracy_score(y_val, y_predval))
    recall_val_g.append(recall_score(y_val, y_predval))
    precision_val_g.append(precision_score(y_val, y_predval))

plt.plot(y_data_g, accuracy_train_g, 'g', label='Training Set')
plt.plot(y_data_g, accuracy_val_g, 'b', label='Validation Set')
plt.ylabel('Accuracy')
plt.xlabel('gamma')
plt.title('Accuracy Scores of Training Set and Validation Set, varying gamma')
plt.legend()
plt.show()

plt.plot(y_data_g, precision_train_g, 'g', label='Training Set')
plt.plot(y_data_g, precision_val_g, 'b', label='Validation Set')
plt.ylabel('Precision')
plt.xlabel('gamma')
plt.title('Precision Scores of Training Set and Validation Set, varying gamma')
plt.legend()
plt.show()
```

```
plt.plot(y_data_g, recall_train_g, 'g', label='Training Set')
plt.plot(y_data_g, recall_val_g, 'b', label='Validation Set')
plt.ylabel('Recall')
plt.xlabel('gamma')
plt.title('Recall Scores of Training Set and Validation Set, varying gamma')
plt.legend()
plt.show()
```

7.5.7 Final model setup

```
# Final model with best  value
model2 = SVC(probability=True, kernel='poly', C=2,  gamma = 0.05)
model2.fit(X_train, y_train)
y_predtr = model2.predict(X_train)
accuracytr = accuracy_score(y_train, y_predtr)
precisiontr = precision_score(y_train, y_predtr)
recalltr = recall_score(y_train, y_predtr)
print(f"training accuracy: {accuracytr}, training precision: {precisiontr}, training recall: {recalltr}")

y_pred = model2.predict(X_val)
accuracy = accuracy_score(y_val, y_pred)
precision = precision_score(y_val, y_pred)
recall = recall_score(y_val, y_pred)
print(f"validation accuracy: {accuracy}, validation precision: {precision}, validation recall: {recall}")

y_predt = model2.predict(X_test)
accuracyt = accuracy_score(y_test, y_predt)
precisiont = precision_score(y_test, y_predt)
recallt = recall_score(y_test, y_predt)
print(f" test accuracy: {accuracyt}, test precision: {precisiont}, test recall: {recallt}")

# Confusion matrix for test set
cm = confusion_matrix(y_test, y_predt)
fig, ax = plt.subplots(figsize=(8, 6))
sns.heatmap(cm, cmap='Blues',annot=True)
ax.set_xlabel('Predicted')
ax.set_ylabel('Reality')
ax.set_title('Confusion Matrix(Test)')
ax.xaxis.set_ticklabels(['Negative', 'Positive'])
ax.yaxis.set_ticklabels(['Negative', 'Positive'])
plt.show()

recall = recall_score(y_test, y_predt)
print(f"recall: {recall}")
```

7.5.8 PCA

```
from sklearn.decomposition import PCA

#Rank the importance of the features with PCA
components = pca.components_
explained_variance = pca.explained_variance_ratio_
feature_importance = np.abs(components) * explained_variance.reshape(-1, 1)
feature_importance = feature_importance.sum(axis=0)
feature_ranking = np.argsort(feature_importance)[::-1]
# Print feature rankings
print("Feature ranking (most important first):")
for rank, index in enumerate(feature_ranking):
    print(f"Rank {rank + 1}: Feature {index} (importance = {feature_importance[index]:.4f})")
```

7.6 Heat Map

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sb

heart = pd.read_csv('heart_disease_uci_2.csv')

# categorical variable into a binary
# cp
heart['cp atypical angina'] = heart['cp'] == 'atypical angina'
heart['cp atypical angina'] = heart['cp atypical angina'].astype(int)

heart['cp non-anginal'] = heart['cp'] == 'non-anginal'
heart['cp non-anginal'] = heart['cp non-anginal'].astype(int)

heart['cp typical angina'] = heart['cp'] == 'typical angina'
heart['cp typical angina'] = heart['cp typical angina'].astype(int)

# restecg
heart['restecg normal'] = heart['restecg'] == 'normal'
heart['restecg normal'] = heart['restecg normal'].astype(int)

heart['restecg st-t abnormality'] = heart['restecg'] == 'st-t abnormality'
heart['restecg st-t abnormality'] = heart['restecg st-t abnormality'].astype(int)

# slope
heart['slope flat'] = heart['slope'] == 'flat'
heart['slope flat'] = heart['slope flat'].astype(int)

heart['slope upsloping'] = heart['slope'] == 'upsloping'
```

```
heart['slope upsloping'] = heart['slope upsloping'].astype(int)

# thal
heart['thal normal'] = heart['thal'] == 'normal'
heart['thal normal'] = heart['thal normal'].astype(int)
heart['thal reversable defect'] = heart['thal'] == 'reversable defect'
heart['thal reversable defect'] = heart['thal reversable defect'].astype(int)

del heart['cp']
del heart['restecg']
del heart['thal']
del heart['slope']

heart.head()

# plotting correlation heatmap
dataplot = sb.heatmap(heart.corr())
# displaying heatmap
plt.show()

plt.savefig('heatmap.png', dpi=300)
```