**Question 1: Read the article and reproduce the results presented in Table-4 using Python modules and packages (including your own script or customised codes). Write a report summarising the dataset, used ML methods, experiment protocol and results including variations, if any.**

## Summary of the Dataset and Features used in each Models

The dataset used by the authors of the research article "Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone" [1] was acquired from a study conducted by Ahmad T. [2] on 299 patients of heart failure in the city of Faisalabad in Pakistan. The patients comprised of 105 women and 194 men aged at least 40 years of age. The data collected during this study includes Age, serum sodium, serum creatinine, gender, smoking, Blood Pressure (BP), Ejection Fraction (EF), anaemia, platelets, Creatinine Phosphokinase (CPK) and diabetes and follow-up time in days [2]. The following table list the variables in the dataset used by the authors.

| Dataset attributes | Attribute type | Independent/dependent |
|---|---|---|
| Age | Continuous | Independent Variables |
| Anaemia | Categorical | |
| Creatine Phosphokinase | Continuous | |
| Diabetes | Categorical | |
| Ejection Fraction | Categorical | |
| High Blood Pressure | Categorical | |
| platelets | Categorical | |
| Serum Creatinine | Categorical | |
| Serum Sodium | Continuous | |
| Sex | Categorical | |
| Smoking | Categorical | |
| Time (days) | Continuous | |
| Death Event | Categorical | Dependent variable |

The dependent variable Death Event is of categorical attribute type where 0 indicates a patient who survived. On the other hand, 1 indicates a patient who did not survive during the following period. There are 203 patients who survived and 96 patients who died during the period of this study from April to December 2015 [2]. This highlights an imbalance in the class Death Event, which may have an impact the Machine Learning (ML) training process, which will be demonstrated later in this report.

The remaining categorical variables are anaemia, diabetes, Ejection Fraction (EF), high blood pressure, platelets, serum creatinine, sex and smoking. These are independent variables collected via blood work or taken from the physician's notes [2]. Age, Creatinine Phosphokinase (CPK), serum sodium and time are on the other hand continuous variables.

The features used in the research paper [1] uses different sets of features for 3 assessments reported in the paper. These are summarized in the table below, showing the ML model used, independent features used and source table from the paper. The research did not include independent variable time in the predictions reported in table 4 but included all the remaining features. Three of the models reported in table 4 were reused to predict only with independent variables serum creatinine and ejection fraction. These models were Random Forest, Gradient Boosting and SVM Radial. The authors selected these two variables for their experiment by doing feature ranking using Random Forest gini impurity reduction [1]. Finally, two more predictions were done using Logistic Regression, firstly with ejection fraction, serum creatinine and follow-up time. Secondly, with all features as shown below.

| **Machine Learning Model** | **Independent Features** | **Source (Table No./[Reference]** |
|---|---|---|
| Random Forest | All clinical features except for follow-up *time* | **table 4 [1]** |
| Decision Tree | | |
| Gradient Boosting | | |
| Linear Regression | | |
| One Rule | | |

| | | |
|---|---|---|
| Artificial Neural Networks (ANN) | | |
| Naïve Bayes | | |
| SVM Radial | | |
| SVM Linear | | |
| K-Nearest Neighbors (KNN) | | |
| Random Forest | Serum creatinine and ejection fraction | **table 9 [1]** |
| Gradient Boosting | | |
| SVM Radial | | |
| Logistic regression | Ejection fraction, serum creatinine and Follow-up *time* | **table 11 [1]** |
| Logistic regression | All features | |

During this experiment, the first set of ML models have been rebuilt as per source table 4 above. The models also have also been built on all features except for follow-up time to reproduce the results obtained by (author) [1]. The dataset for heart failure clinical records has first been loaded to perform any pre-processing on the data before building the models. As part of this step, blank rows and fields have been checked, of which none were identified. The original dataset has been split between independent features, stored in data frame hfcr_ed_X, and dependent feature, stored in data frame hfcr_y. To match the features required for the models in table 4 of the research paper [1], time has been removed from the independent variables data frame hfcr_ed_X. No further pre-processing has been done on this dataset before producing the predictions in table 4, hence the models were built without standardization and class rebalancing for the training dataset.

## Machine Learning Models, Experiment Protocols and Assessment of Results

Each of the ML models rebuilt is described in this section, with details on model selection, hyper-parameters, grid search, training, testing and results. For this experiment the same models built in table 4 of the research paper [1] has been reproduced. A combination of R script and Lua script were originally used by the authors for the models in the paper. The models were reproduced using Python programming language by taking advantage of the various packages for ML that are prebuilt and available to use in the Scikit-learn libraries.

Before describing each model, the following are to be noted as it applies to more than one model. All the models have training and test split of 80 percent to 20 percent. In addition, 4 of these models namely ANN, KNN, SVM radial and SVM linear further split the 80 percent training set to 60 percent for training and 20 percent for validation. The validation set has been used to assess performance of hyper-parameters during the tuning phase which was performed by a custom grid search. The process for splitting the training and test datasets by the authors in the research paper [1] is based on custom code to shuffle the rows of the dataset before splitting, this ensures that the dataset is randomise at each iteration of the model running. Similarly, for this experiment, each time the model is run the dataset in split on randomised selection, however this is done using the python package Train Test Split from the Python Scikit-learn library Model Selection. This package performs similar randomisation on selection of the training and test dataset. This slight variation in approach could explain some of the differences between assessment results of this experiment and the research paper.

The models built were assessed using the metrics Matthews Correlation Coefficient, F1-Score, Accuracy, True Positive (TP) rate, True Negative (TN) rate, Precision-Recall (PR) Area Under Curve (AUC) and Receiver Operator Characteristics (ROC) AUC. Furthermore, the hyper-parameter grid search was assessed using Matthews Correlation Coefficient. To replicate this model assessment process and grid search, a custom python function has been built to return the metric used to measure model performance which is called by each ML model code section to return the performance metrics. The hyper-parameter tuning was performed in the same section where the grid input is passed into a loop to perform a grid. For consistency, instead of using the python package GridSearchCV from the Python library Model Selection from Scikit-learn, a custom Python grid search has been used to perform the same process by using Matthews Correlation Coefficient (MCC) to select the best hyper-parameter.

Each of the models in the research paper were run 100 times to collect the results of each run and report on the mean of the set of results. The same approach has been done on this experiment to get a randomised prediction for each of the 100 runs of each model. To achieve this in Python a similar loop function has been used and the Train Test Split method random state was not set which means a new randomised training and test samples is selected at each iteration. Next is a review of each model built in detail along with results assessment.

## Linear Regression Model

The Linear Regression ML model has been built with default settings from scikit-learn Linear Model library for linear regression. The dataset was randomly split for training at 80% and testing at 20%. The random split ran for 100 iterations and the mean results of the metrics used for model assessment returned the performance reported in the table below. The Python Linear Regression Model results listed against the results of the research paper. The linear regression results were converted to 0 and 1 to mimic a classifier. Hence, if the prediction results were less that 0.5, prediction were classified as 0 for survived, and more than or equal to 0.5 were classified as dead.

| Method: **Linear Regression** | MCC | F1 Score | Accuracy | TP rate | TN rate | PR AUC | ROC AUC |
|---|---|---|---|---|---|---|---|
| Python Model | 0.357 | 0.488 | 0.74 | 0.402 | 0.902 | 0.463 | 0.652 |
| Research Paper | 0.332 | 0.475 | 0.73 | 0.394 | 0.892 | 0.495 | 0.643 |
| **Difference** | -0.025 | -0.013 | -0.01 | -0.008 | -0.01 | 0.032 | -0.009 |

The differences in prediction are negligible given the difference obtained between the two implementations of the model. The change is sample selection for the training and test split process generates differences in the predictions, however these are expected difference as opposed to anomalies. Each time the Python model is run, it will iterate 100 times and give a slightly different number, however still very close to the results reported in the research paper.

## Naïve Bayes Model

The naïve Bayes model used for this experiment is the Gaussian Naïve Bayes package available form Scikit-learn library. Same protocols as per the linear regression model were applied, hence the standard package was used with default parameters.

| Method: **Naïve Bayes** | **MCC** | **F1 Score** | **Accuracy** | **TP rate** | **TN rate** | **PR AUC** | **ROC AUC** |
|---|---|---|---|---|---|---|---|
| Python Model | 0.223 | 0.335 | 0.698 | 0.242 | 0.921 | 0.402 | 0.581 |
| Research Paper | 0.224 | 0.364 | 0.696 | 0.279 | 0.898 | 0.437 | 0.587 |
| **Difference** | 0.001 | 0.029 | -0.002 | 0.037 | -0.023 | 0.035 | 0.006 |

There are very small differences between the research paper and the package built for this review. These can be attributed to the randomisation of training and testing dataset split.

## K-Nearest Neighbors Model

The K-Nearest Neighbors model build for the experiment has been designed slightly differently from the previous models. Firstly, the training dataset is further split to create a validation set for hyper-parameter tuning. The dataset is split 60 percent for training, 20 percent for validation and 20 percent for testing. The overall predictions iterate 100 times, with the data split recurring at each iteration. The number of neighbors parameter (k) is identified through a custom grid search assess by Matthew's Correlated Coefficient scores. The grid search is run 100 times.

With varying setting of K ranging from 1 to 100. The best K value is selected to run the predictions. This hyper-parameter selection is also repeated for each of the 100 iterations of the model's prediction. In this experiment the value of K is passed to the main model after the grid search is complete, hence the K value may change based on the randomness of the training-validation-test split of the data. The results also differ according with the research paper due to this process. The mean of the 100 runs of the model is reported in the table below. During the experiment it is observed that even though different k hyper-parameter may be selected dynamically based on the grid search, the overall results slightly differ from the ones reported in the paper as shown below.

| Method: **K-Nearest Neighbors** | **MCC** | **F1 Score** | **Accuracy** | **TP rate** | **TN rate** | **PR AUC** | **ROC AUC** |
|---|---|---|---|---|---|---|---|
| Python Model | -0.014 | 0.11 | 0.645 | 0.086 | 0.909 | 0.324 | 0.498 |
| Research Paper | -0.025 | 0.148 | 0.624 | 0.121 | 0.866 | 0.323 | 0.493 |
| **Difference** | -0.011 | 0.038 | -0.021 | 0.035 | -0.043 | -0.001 | -0.005 |

Python Model hyper-parameter settings: K = 3. This setting has produced the above results for the python model. Both the K param eter and the results may defer slightly during iteration of the model due to randomness of both training-testing data split and K neare st Neighbors algorithm. The research paper selected K = 3.

## Decision Tree

The Decision Tree Classifier model follows the same iterative process as per Linear Regression and Naïve Bayes. The model uses an 80 per cent to 20 percent training and test dataset split. The model used in this experiment is the Decision Tree Classifier package available from Scikit-learn, the default settings used for the prediction match. The main default parameters were criterion set to gini, max_depth set to none, max features set to none, minimum sample split set to 2, minimum samples leaf set to 1 and splitter set to best.

| Method: **Decision Tree** | **MCC** | **F1 Score** | **Accuracy** | **TP rate** | **TN rate** | **PR AUC** | **ROC AUC** |
|---|---|---|---|---|---|---|---|
| Python Model | 0.25 | 0.48 | 0.675 | 0.484 | 0.764 | 0.407 | 0.624 |
| Research Paper | 0.376 | 0.554 | 0.737 | 0.532 | 0.831 | 0.506 | 0.681 |
| Difference | 0.126 | 0.074 | 0.062 | 0.048 | 0.067 | 0.099 | 0.057 |

The authors of the research paper did not specify any hyper-parameter tuning for the Decision Tree model. The differences observed here are due to the randomness of the data splitting and tuning of the Decision Tree used by the authors not reported in the research paper. However, the differences are not significant.

## Gradient Boosting Model

The Gradient Boosting model used in the experiment follows the same training and testing split as for linear regression model of 80 per cent training set and 20 percent testing set. The hyper-parameters tuned for the model are maximum depth of 2, learning rate of 1, number of estimators is set to 2. These were used by the authors and also as reported in the code supplied with the research [1]. These settings produced matched results between this experiment and the reported results with a small variation due to randomisation of training and testing dataset split.

| Method: **Gradient Boosting** | **MCC** | **F1 Score** | **Accuracy** | **TP rate** | **TN rate** | **PR AUC** | **ROC AUC** |
|---|---|---|---|---|---|---|---|
| Python Model | 0.388 | 0.549 | 0.744 | 0.511 | 0.855 | 0.48 | 0.683 |
| Research Paper | 0.376 | 0.527 | 0.738 | 0.477 | 0.86 | 0.594 | 0.754 |
| Difference | -0.012 | -0.022 | -0.006 | -0.034 | 0.005 | 0.114 | 0.071 |

## One Rule Model

The One Rule classifier implementation in Python was possible from the extension *mlextend* that has been installed for this classification exercise. This package has a library for classification containing the One Rule Classifier algorithm. In addition to the package installation, to perform classification using One Rule the continuous variables must be discretised. The scikit-learn pre-processing library package KBinsDiscretizer was used to discretise all the features before running the classification. The continuous variable was split into bins of 10 groups uniformly using ordinal encoding for this experiment. However, no details of this pre-processing step were provided by the author of the research paper. Training and Test spit were 80 percent to 20 per cent, iterated 100 times through the classifier. The mean of the results was reported in the research paper and this experiment as shown below.

| Method: **One Rule** | MCC | F1 Score | Accuracy | TP rate | TN rate | PR AUC | ROC AUC |
|---|---|---|---|---|---|---|---|
| Python Model | 0.268 | 0.344 | 0.706 | 0.254 | 0.93 | 0.417 | 0.592 |
| Research Paper | 0.319 | 0.465 | 0.729 | 0.383 | 0.892 | 0.482 | 0.637 |
| Difference | 0.051 | 0.121 | 0.023 | 0.129 | -0.038 | 0.065 | 0.045 |

The results had variations due to randomisation of training and test split of the dataset. However, some of the variation could be related to the discretisation process.

## Random Forest Method

The Random Forest method used by the researchers produced similar results to the Python version in this experiment. Default parameters were used for the classifier and same train-test split protocol as for the Linear Regression model described earlier. The mean results have been listed in the table below. Noting very small differences for all results except for PR AUC and ROC UAC which has a slightly higher difference. However, it can be concluded that these differences are due to the random split of data within the modelling process.

| Method: **Random Forest** | MCC | F1 Score | Accuracy | TP rate | TN rate | PR AUC | ROC AUC |
|---|---|---|---|---|---|---|---|
| Python Model | 0.376 | 0.533 | 0.742 | 0.48 | 0.868 | 0.469 | 0.674 |
| Research Paper | 0.384 | 0.547 | 0.74 | 0.491 | 0.864 | 0.657 | 0.8 |
| Difference | 0.008 | 0.014 | -0.002 | 0.011 | -0.004 | 0.188 | 0.126 |

## Artificial Neural Network (ANN) Model

The Artificial Neural Network model developed was trained on 60 percent of the dataset. Another 20 percent of the data set was used for validation of the grid search used to identify the best hyper parameters for number of units under the hidden later. The last 20 percent of the dataset has been used for testing. The solution ran 100 times performing a grid search at each iteration and then predicting using the best value of parameter “number of units” for each iteration. The number of units tested were 5, 10, 25, 50, 75, 100, 125, 150, 175, 200, 225, 250, 275, 300. The scores were averaged using mean to calculate the outcome. According to the research paper, only 1 hidden layer was parametrized during training, validation and testing. The package used in Python was the Multi Later Perceptron from Scikit-learn library with the same protocol and using the defaults settings except for number of units. The only result similar to the research paper were Accuracy and True Positive rate. The remaining results had a high variance. Some of these could be attributed to training, validation and testing dataset random selection. However, this would not account for the whole variance. Given the limited information provided by the paper on the hyper parameter tuning process, it can only be assumed that some tunings were not reported that could explain the remaining variance.

| Method: **Artificial Neural Network** | **MCC** | **F1 Score** | **Accuracy** | **TP rate** | **TN rate** | **PR AUC** | **ROC AUC** |
|---|---|---|---|---|---|---|---|
| Python Model | -0.006 | 0.239 | 0.509 | 0.481 | 0.513 | 0.314 | 0.497 |
| Research Paper | 0.262 | 0.483 | 0.68 | 0.428 | 0.815 | 0.75 | 0.559 |
| Difference | 0.268 | 0.244 | 0.171 | -0.053 | 0.302 | 0.436 | 0.062 |

## SVM Radial Model

The Support Vector Machine (SVM) Radial ML model followed similar training, validation and testing split as for ANN. The model also used a grid search to identify the C parameter that would return the best MCC score. The C parameters used in the grid search were 0.001, 0.01, 0.1, 1 and 10. The model ran 100 times cycling through each C parameter during each of the 100 iterations to find the best C value. The experiment produced the best MCC with C value of 10. The C value can change due to randomness of training and validation set. The kernel used for the SVM model was radial.

The results returned where not close to the research paper except for accuracy and True Negative rate scores. The C value reported by the research was also 10, hence the difference can only be link to how the data has been randomised and used for modelling.

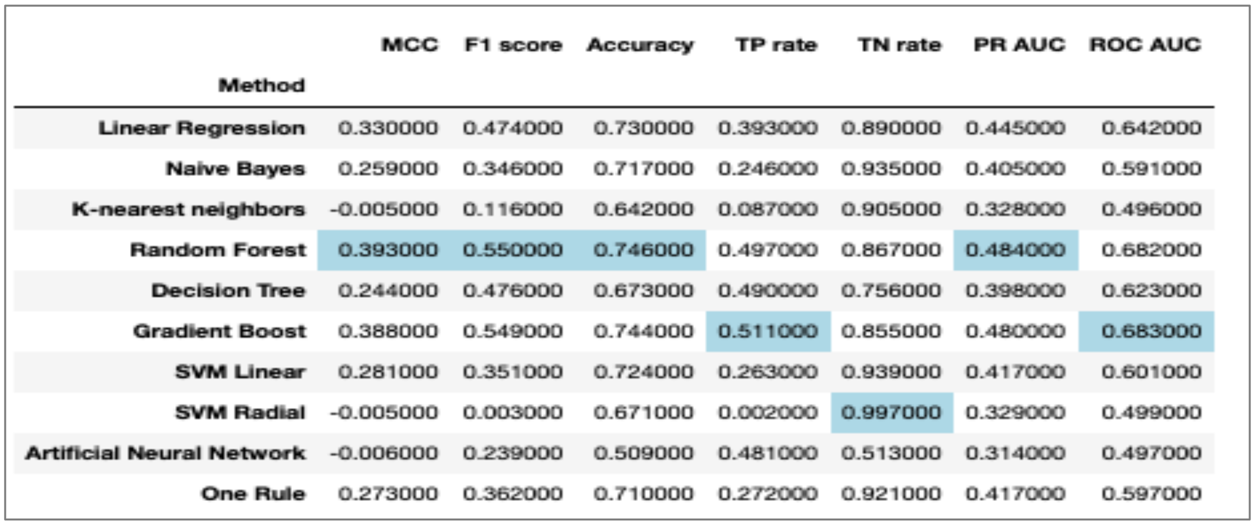
| Method: **SVM Radial** | **MCC** | **F1 Score** | **Accuracy** | **TP rate** | **TN rate** | **PR AUC** | **ROC AUC** |
|---|---|---|---|---|---|---|---|
| Python Model | -0.01 | 0.003 | 0.679 | 0.002 | 0.996 | 0.319 | 0.499 |
| Research Paper | 0.159 | 0.182 | 0.69 | 0.122 | 0.967 | 0.587 | 0.749 |
| Difference | 0.169 | 0.179 | 0.011 | 0.12 | -0.029 | 0.268 | 0.25 |

## SVM Linear model

The Linear Support Vector Machine (SVM) ML Model built followed the same protocol as for SVM radial for training, validation and testing. The grid search also had the same parameter. The Scikit-learn package LinearSVC was used for the development of this model which is equivalent to the model used by the authors of the research paper. The results produced two similar metrics, namely TN rate with a difference of only 0.042 and accuracy with a difference of -0.027.

| Method: **SVM Linear** | **MCC** | **F1 Score** | **Accuracy** | **TP rate** | **TN rate** | **PR AUC** | **ROC AUC** |
|---|---|---|---|---|---|---|---|
| Python Model | 0.261 | 0.333 | 0.711 | 0.247 | 0.939 | 0.414 | 0.593 |
| Research Paper | 0.107 | 0.115 | 0.684 | 0.072 | 0.981 | 0.594 | 0.754 |
| Difference | -0.154 | -0.218 | -0.027 | -0.175 | 0.042 | 0.18 | 0.161 |

Summary of overall results from the built models with highlighted best scores is shown below. The highest scores do differ from the research paper, hence hard to assess differences. Nevertheless, Random Forest was the best performing models in this experiment and the research model with 3 plus top scores. ANN and Decision Tree did not score any best results for our experiment but in the research paper Decision Tree has the best F1 Score of 0.554 and True Positive rate of 0.532.  ANN produced the best PR AUC score of 0.750. Gradient Boost model on the built model produced highest scores for TP rate and ROC AUC. SVM models were similar with highest scores for TN rate scores.



| Method | MCC | F1 score | Accuracy | TP rate | TN rate | PR AUC | ROC AUC |
|---|---|---|---|---|---|---|---|
| Linear Regression | 0.330000 | 0.474000 | 0.730000 | 0.393000 | 0.890000 | 0.445000 | 0.642000 |
| Naive Bayes | 0.259000 | 0.346000 | 0.717000 | 0.246000 | 0.935000 | 0.405000 | 0.591000 |
| K-nearest neighbors | -0.005000 | 0.116000 | 0.642000 | 0.087000 | 0.905000 | 0.328000 | 0.496000 |
| Random Forest | 0.393000 | 0.550000 | 0.746000 | 0.497000 | 0.867000 | 0.484000 | 0.682000 |
| Decision Tree | 0.244000 | 0.476000 | 0.673000 | 0.490000 | 0.756000 | 0.398000 | 0.623000 |
| Gradient Boost | 0.388000 | 0.549000 | 0.744000 | 0.511000 | 0.855000 | 0.480000 | 0.683000 |
| SVM Linear | 0.281000 | 0.351000 | 0.724000 | 0.263000 | 0.939000 | 0.417000 | 0.601000 |
| SVM Radial | -0.005000 | 0.003000 | 0.671000 | 0.002000 | 0.997000 | 0.329000 | 0.499000 |
| Artificial Neural Network | -0.006000 | 0.239000 | 0.509000 | 0.481000 | 0.513000 | 0.314000 | 0.497000 |
| One Rule | 0.273000 | 0.362000 | 0.710000 | 0.272000 | 0.921000 | 0.417000 | 0.597000 |

*The scores in the table above are rounded to 3 decimal places. The figures may not be similar the ones reported in previous section due to randomisation of the modelling process and generated during multiple iterations.*

**Question 2: Design and develop your own ML solution for this problem. The proposed solution should be different from all approaches mentioned in the provided article. This does not mean that you must have to choose a new ML algorithm. You can develop a novel solution by changing the feature selection approach or parameter optimisations process of used ML methods or using different ML methods or different combinations of them. This means, the proposed system should be substantially different from the methods presented in the article but not limited to only change of ML methods. Compare the result with reported methods in the article. Write a technical report summarising your solution design and outcomes.**

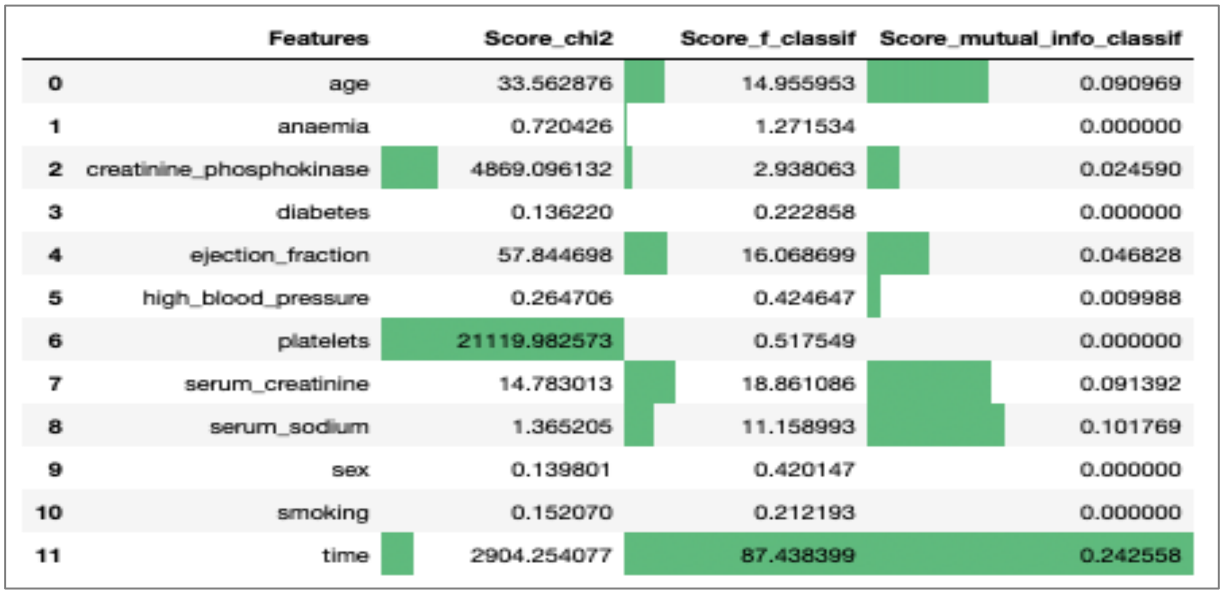
The design decisions for the Machine Learning the model were based on an analysis of how the models in the research paper were built, to identify gaps and areas of improvement. Some issues identified are:

- The dataset had an imbalance class which could have decreased training quality
- The dataset was not scale before feeding the machine learning models
- The grid search technique used was limited

The above were addressed by using prebuilt packages from the Scikit-learn library and solutions were applied to the modelling pipeline to achieve better results. In the case of class imbalance, a verification step was done to validate possible improvements. As for grid search, an iterative approach was taken to select to grid search input parameters. Then decision was made to leverage a stack ensemble to get a better prediction by combining some of the Machine Learning models with hyper-parameter tuning. The stack ensemble included 5 base learners and one meta-learner. This will be further discussed in later sections. This approach is different from the research paper models as is uses a combination of approaches as describe previously to provide a Machine Learning model that is not only different in design but also provide differentiating performance improvements by combining various ML algorithms. The following sections details the steps that were taken to develop the ML model.

## Feature Selection

The table below shows a side-by-side comparison of the scikit-learn SelectKBest feature selection results on the heart failure dataset using three score functions. The method used are classification scores chi2, ANOVA F-value and mutual information for classification. The results from ANOVA F-value and mutual information returned similar results, selecting time, serum sodium, serum creatinine, ejection fraction and age as top 5 features in the dataset. Chi2 on the other hand, selected platelets, creatine phosphokinase, time, ejection fraction and age as top 5 features. The feature selection output illustrated below.



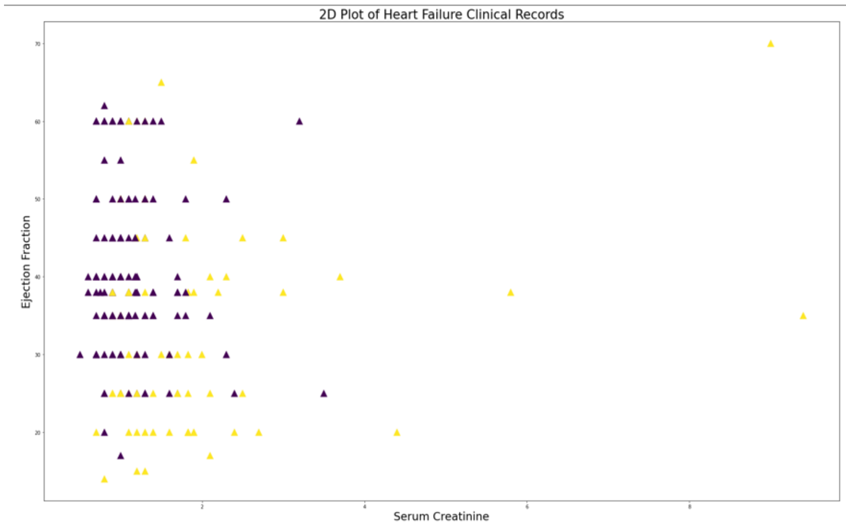
| | Features | Score_chi2 | Score_f_classif | Score_mutual_info_classif |
|---|---|---|---|---|
| 0 | age | 33.562876 | 14.955953 | 0.090969 |
| 1 | anaemia | 0.720426 | 1.271534 | 0.000000 |
| 2 | creatinine_phosphokinase | 4869.096132 | 2.938063 | 0.024590 |
| 3 | diabetes | 0.136220 | 0.222858 | 0.000000 |
| 4 | ejection_fraction | 57.844698 | 16.068699 | 0.046828 |
| 5 | high_blood_pressure | 0.264706 | 0.424647 | 0.009988 |
| 6 | platelets | 21119.982573 | 0.517549 | 0.000000 |
| 7 | serum_creatinine | 14.783013 | 18.861086 | 0.091392 |
| 8 | serum_sodium | 1.365205 | 11.158993 | 0.101769 |
| 9 | sex | 0.139801 | 0.420147 | 0.000000 |
| 10 | smoking | 0.152070 | 0.212193 | 0.000000 |
| 11 | time | 2904.254077 | 87.438399 | 0.242558 |

The selected features were grouped into two sets for validation. The first set was a group of 5 features selecting time, serum sodium, serum creatinine, ejection fraction and age. The second set being was similar to the three features selected the by Logistic Regression ML model in the research paper [1], selecting only serum creatinine, ejection fraction and time. Based on the prediction results, the latter performed much better across all metrics, including a high score of 94 percent for True Positive rate.

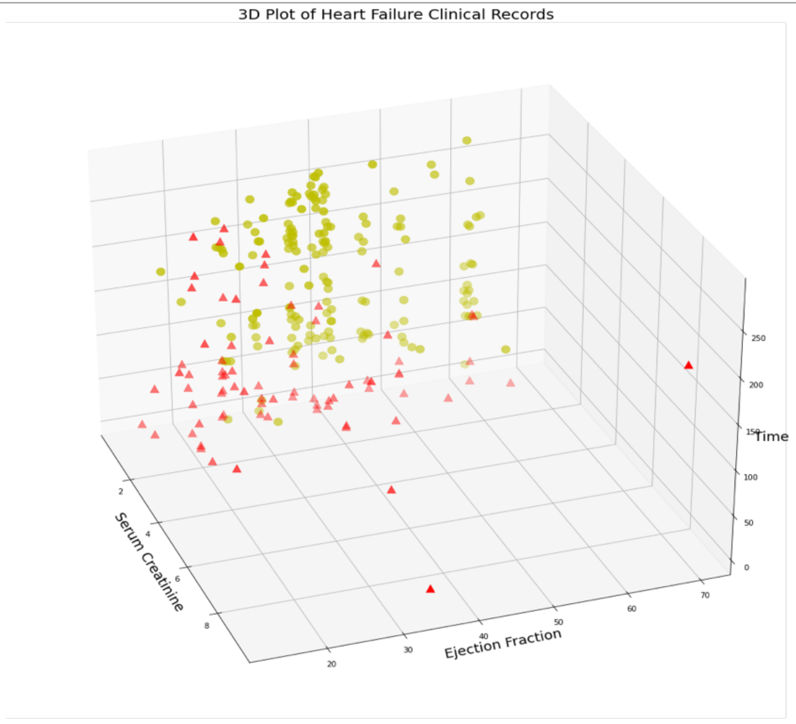
The feature selection process was performed only on the training dataset after the test dataset has been split. This ensures the test dataset is not seen before predictions on the test data.

## Visualisation of Selected Features

This section briefly visualizes the selected features derived from the feature selection process. The dataset being high-dimensional, only a few keys data features has been targeted for illustration purposes. The next 2-dimensional plot shows that the dataset has two distinct groups with some overlaps. Lower right for class 1 and upper left for class 0.



2-dimensional plot of the Heart Failure dataset with serum creatinine on the x axis and ejection fraction on the y axis. Purple markers are cases that survived and yellow markers are cases who did not survive



3-dimensional plot of the Heart Failure dataset with serum creatinine, ejection fraction and follow-up time on the 3rd axis. Yellow markers are cases that survived and red markers are cases who did not survive

The second scatter plot is a 3-dimensional illustration of serum creatinine, ejection fraction and time. This view of the dataset shows that the data set is non-linear and has a curved boundary between class 1 and 0.

## Imbalance Class

The dataset as mentioned in previous sections has class imbalance. The number of samples for Death Event equal to zero is much more than the Death Event samples equal to one, 203 records against 96 respectively. A process was adopted to validate whether synthetic samples in the training set would improve the predictions of unseen data. To achieve this a logistic regression model was fitted to the training set before applying Synthetic Minority Over-sampling Technique (SMOTE). Then validate any improvements on True Positive prediction results by measuring improvements to the recall metric.
The table below shows the impact on SMOTE on the dataset using logistic regression modelling technique. An improvement of 8 percent is observed in test set predictions recall. Base on this SMOTE has been applied to the modelling pipeline.

| | Training dataset | Training size of minority class | Testing dataset |
|---|---|---|---|
| Recall before SMOTE | 69% | 71 | 64% |
| Recall After SMOTE | 80% | 138 | 72% |

*These results can be referenced in section 3 of the code file*

## Standardisation

The training and testing dataset have been standardize for the predictions. The method used for standardising the data with the Scikit-learn standard scaler method. This has been applied to the dataset after resolving the training dataset class imbalance.

## Hyper-parameter Tuning

The hyper-parameter tuning of the base models selected for the stack ensemble method was done through grid search. The GridSearchCV package from Scikit-learn library was used to identify the best parameters for each Machine Learning models. The best parameters were identified in section 4 of the source code. These were passed into the ensemble model, which will be discussed in the next section.

The base learners included in the ensemble model were K-Neighbors Classifier (KNC), Decision Tree Classifier (DTC), Logistic Regression CV, Gradient Boosting Classifier and Support Vector Classifier (SVC). The table below highlights the parameters processed by the grid search for each base learner and the best results. These parameters are identified and passed in the ensemble dynamically, hence there might be slight variations in the parameters set for each base learners once the prediction model is run, this ensures a continuous assessment of parameters based on input data to provide best possible parameter input for the base learner models at each iteration. The table below summarised this process.

| **Base learners** | **GridSearchCV input parameters** | **<u>Best Parameters</u> identified by GridSearchCV** | **Accuracy** |
|---|---|---|---|
| K-Neighbors Classifier | n_neighbors: range (1,18,2)<br>weights: ['uniform', 'distance']<br>metric': ['euclidean', 'manhattan', 'minkowski', 'chebyshev'] | Metric = manhattan<br>n_neighbors = 13<br>weights = distance | 88% |
| Decision Tree Classifier | criterion: ['entropy', 'gini']<br>max_depth: [2,3,4,5,6,7,8,9,10]<br>max_leaf_nodes: [2,3,4,5] | max_depth = 3<br>max_leaf_nodes = 4 | 84% |
| Logistic Regression CV | solver: ['newton-cg', 'lbfgs', 'liblinear',' sag', 'saga']<br>Cs: [0.0001,0.001,0.01,0.1,1,10]<br>penalty: ['l1','2','elasticnet'] | Penalty = l1<br>Solver = liblinear | 77% |
| Gradient Boosting Classifier | max_depth: [2,3,4,5,6,7,8,9,10]<br>n_estimators: [2,10,50,100]<br>learning_rate: [1,2,3] | learning_rate = 1<br>max_depth = 5<br>n_estimators=10 | 86% |

| | | | |
|---|---|---|---|
| Support Vector Classifier | kernel: ['rbf'] gamma: [1e-3, 1e-4] C: [1, 10, 100, 1000] kernel: ['linear'] C: [1, 10, 100, 1000] | C=1000 gamma=0.001 | 82% |

Base on accuracy scores K-Neighbors Classifier returned the best score with parameters set to number of neighbors equal to 13, metric for distance measurement Manhattan and weights based on distance. The worse performing base learner with the GridSearchCV is Logistic Regression Cross-Validation which scored 77 percent accuracy with L1 regularizer and solver set to liblinear. SVC, Gradient Boost Classifier and Decision Tree Classifier perform well, giving accuracy sores in the 80-90 percent range. This ensured a good baseline for the ensemble described in the next section.

**Machine Learning Model**

The method used for prediction in this experiment is a stacked ensemble method that is made up of the base learners describe in the previous section and a meta learner that is built on top of the base learners. This type of ensemble is non-generative, that is it learns how to best combine the base learners without having an impact on the base learners [4].

The ensemble model is represented in section 5 of the code file. The code was based on the stack ensemble example in [3] and [4]. There are pre-processing done on the dataset before prediction. Firstly, the dataset is split for training at 80 percent and testing at 20 percent. Then SMOTE is applied to the training dataset after the data split. Finally, the dataset is standardised by applying Scikit-learn package standard scaler to both training and test dataset. For the final model, independent features used with the model are time, serum creatinine and ejection fraction. This same model was run with the top 5 features of the feature selection process, however the top 3 features return the best observed results.

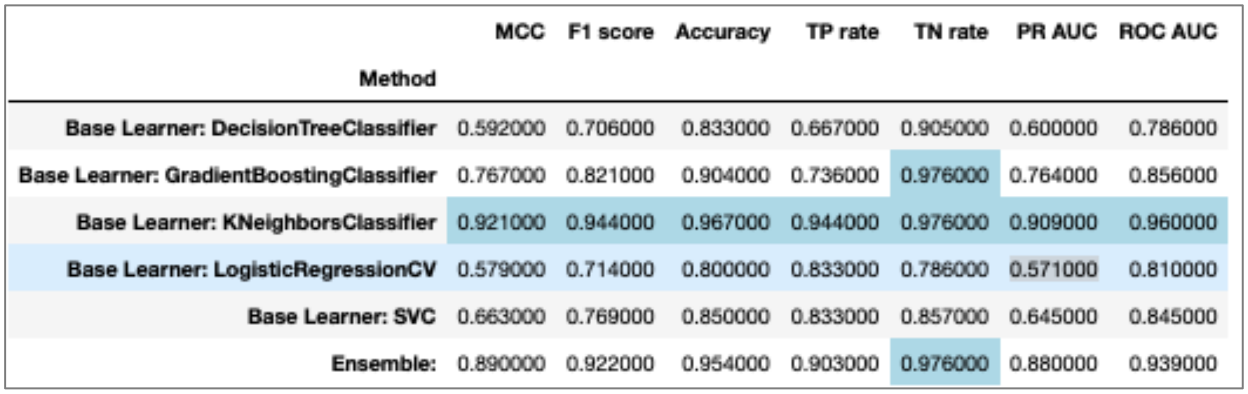
After pre-processing, each base-learner's best parameters were stored in a data frame after applying GridSearchCV similarly to what was detailed in the hyper-parameter section of this report. A function for GridSearchCV has been built to efficiently perform the grid search and report the model with the best estimator for the data frame used in the base-learner prediction process.

The base-learners were trained using Scikit-learn KFold cross-validator on the training dataset. This was manually tuned to eight splits to get the best results from the ensemble model. Split greater than eight returned less performance in prediction. The base-learners accuracy scores are K-Neighbors Classifier 97 percent, Decision Tree Classifier 83 percent, Logistic Regression CV 80 percent, Gradient Boosting Classifier 90 percent, Support Vector Classifier 85 percent.

The meta-learner used for the stacked ensemble was Logistic Regression. This was used to for fitting and prediction of the ensemble model. The performance of the ensemble model was 95 percent. The next section breaks down the prediction results into more metrics further explain the assessment metrics in the performance review and assist in for comparison to the research paper.

**Overall Results**

For this report the model was run three times and the mean of the results can be seen below. K-Neighbors Classifier is the overall best performing model in the stacked base learners. This model was added to the stack last, hence this raised the performance significantly. Excluding this base learner, the ensemble does provide better metrics than what the authors reported in the research paper for all except for True Positive rate. With the introduction of K-Neighbors Classifier the True Positive rate improved. The mean of the metrics below shows consistent high values for MCC, F1-score, Accuracy, TP rate, TN rate, PR AUC and ROC AUC as shown in the table below. Highlighted cells are best scores.



| Method | MCC | F1 score | Accuracy | TP rate | TN rate | PR AUC | ROC AUC |
|---|---|---|---|---|---|---|---|
| Base Learner: DecisionTreeClassifier | 0.592000 | 0.706000 | 0.833000 | 0.667000 | 0.905000 | 0.600000 | 0.786000 |
| Base Learner: GradientBoostingClassifier | 0.767000 | 0.821000 | 0.904000 | 0.736000 | 0.976000 | 0.764000 | 0.856000 |
| Base Learner: KNeighborsClassifier | 0.921000 | 0.944000 | 0.967000 | 0.944000 | 0.976000 | 0.909000 | 0.960000 |
| Base Learner: LogisticRegressionCV | 0.579000 | 0.714000 | 0.800000 | 0.833000 | 0.786000 | 0.571000 | 0.810000 |
| Base Learner: SVC | 0.663000 | 0.769000 | 0.850000 | 0.833000 | 0.857000 | 0.645000 | 0.845000 |
| Ensemble: | 0.890000 | 0.922000 | 0.954000 | 0.903000 | 0.976000 | 0.880000 | 0.939000 |

Best results from the research paper [1] were:

| Method: | MCC | F1 Score | Accuracy | TP rate | TN rate | PR AUC | ROC AUC |
|---|---|---|---|---|---|---|---|
| Logistic Regression | 0.616 | 0.719 | 0.838 | 0.785 | 0.86 | 0.617 | 0.822 |

*References:*

[1] D. Chicco, G. Jurman, “Machine Learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone”, *BMC Medical Informatics and Decision Making*, vol. 20, no. 16, 2020

[2] T. Ahma, A. Munir, S. H. Bhatti, M. Aftab, M. A. Raza, “Survival analysis of heart failure patients: A case study”, *PLOS ONE*, 12(7): e0181001, Jul., 2017.

[3] Kyriakides G and Margaritis K (2019) Hands-On Ensemble Learning with Python, 1st edn, Packt Publishing, UK

[4] Kyriakides G and Margaritis (2019) Stacking for classification, [source code], https://github.com/PacktPublishing/Hands-On-Ensemble-Learning-with-Python/blob/master/Chapter04/stacking_classification.py