Group 39 : Heart Disease AI

Module Name: Artificial Intelligence

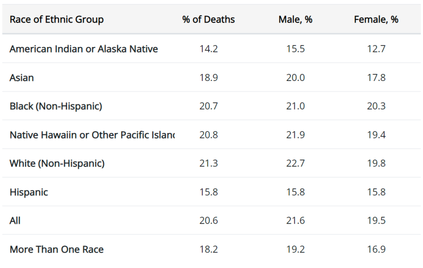
Date: 19/01/2023

Abstract - Heart diseases are prevalent in today's world. Many patients were wrongly diagnosed and, as a result, given wrong treatments which potentially resulted in death. We used a dataset based on Hungarian data. Data was properly aligned and thoroughly pre-processed. 3 machine learning algorithms were chosen to classify individuals. This project focused on building models that perfectly encapsulate the understanding of heart disease symptoms and give an accurate diagnosis to individuals.

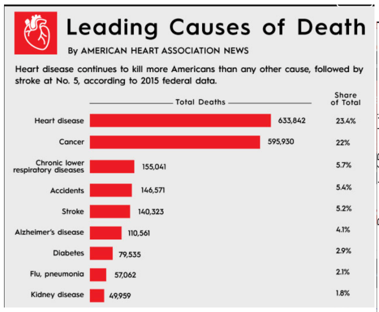
Index Terms - classification, heart, regression, ensemble, knn, diagnosis, visualisation

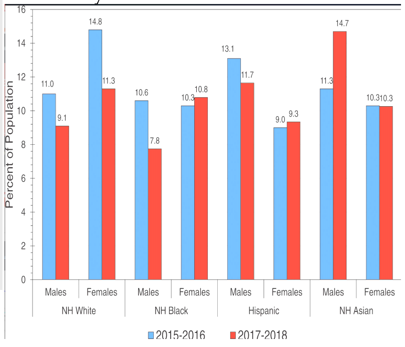
1. INTRODUCTION

Heart diseases are prevalent in today's world. Globally, it was estimated that in 2020, about 244 million people were living with ischemic heart disease (IHD)[1], with it being more prevalent in males. About 700,000 people in the United States alone died from heart diseases in 2020[2]. Despite heart diseases being very common, misdiagnosis is recurrent. The rates of heart failure misdiagnosis ranged from 16.1% in hospital settings to 68.5% when general practitioners referred patients to specialist settings. The most common cause for misdiagnosis was chronic obstructive pulmonary disease (COPD)[3]. Misdiagnoses are often easy to miss and are very difficult to detect. It is underemphasized in today's world.



Race of Ethnic Group	% of Deaths	Male, %	Female, %
American Indian or Alaska Native	14.2	15.5	12.7
Asian	18.9	20.0	17.8
Black (Non-Hispanic)	20.7	21.0	20.3
Native Hawaiin or Other Pacific Islan	20.8	21.9	19.4
White (Non-Hispanic)	21.3	22.7	19.8
Hispanic	15.8	15.8	15.8
All	20.6	21.6	19.5
More Than One Race	18.2	19.2	16.9





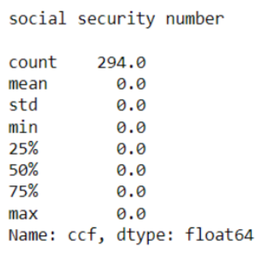
Our main objective is to build algorithms using machine learning to accurately classify the probability of an individual having heart disease. The machine algorithms that we use will learn from the training dataset and hyperparameter tuning (Experience E) to increase efficacy in heart disease diagnosis (Task T). This will then be illustrated with our confusion matrix and classification reports, where we will mainly use accuracy as our scoring method (Measure P). In order to properly achieve this supervised learning task, we need to select appropriate features from our original data which will help our algorithms learn to predict heart disease. The data that we utilise will include many variables based on Hungarian data. We had to deal with a rather fair amount of missing and redundant data. The dataset also included many variables that seem to have little to no association with heart diseases. In spite of this, we managed to arrange and clean the data, making it relevant and useful to our task at hand. The data was thoroughly pre-processed and was split into the training data, validation data and the test data, which was used to test between our final models after hyperparameter tuning.

We had a total of 25 inputs (the heart-disease.names file contains further information) to deal with in our machine learning task after preprocessing was completed, namely: ***age*** *(in years)*, ***sex*** *(male or female)*, ***painloc*** *(chest pain location)*, ***painexer*** *(pain provoked by exertion)*, ***relrest*** *(pain relieved after rest)*, ***cp*** *(type of chest pain)*, ***trestbps*** *(resting systolic blood pressure in mmHg)*, ***htn*** *(presence of hypertension)*, ***chol*** *(serum cholesterol level)*, ***fbs*** *(fasting blood sugar > 120 mg/dl)*, ***restecg*** *(resting electrocardiogram results)*, ***prop*** *(beta blocker used during exercise ECG)*, ***nitr*** *(nitrates used during exercise ECG)*, ***pro*** *(calcium channel blocker used during exercise ECG)*, ***thaldur*** *(duration of exercise test in minutes)*, ***met*** *(metabolic equivalent of task)*, ***thalach*** *(maximum heart rate achieved)*, ***thalrest*** *(resting heart rate)*, ***tpeakbps*** *(peak exercise systolic blood pressure)*, ***tpeakbpd*** *(peak exercise diastolic blood pressure)*, ***trestbpd*** *(resting diastolic blood pressure)*, ***exang*** *(exercise induced angina)*, ***oldpeak*** *(ST depression induced by exercise relative to rest)*, ***rldv5*** *(height at rest)*, ***rldv5e*** *(height at peak exercise)*. There is one output feature: ***num*** *(diagnosis of heart disease based on 50% diameter narrowing in any major blood vessel of the heart)*.

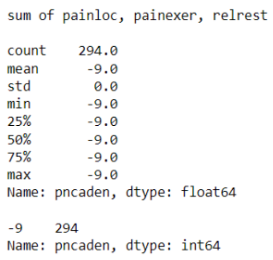
We have collaboratively selected 3 main machine learning algorithms that will aid in our classification models, which included an ensemble stack. Each of these algorithms will classify with their relative degrees of accuracy. Logistic regression and K-Nearest Neighbours will be used as well.
We desire to build a model that will correctly and efficiently diagnose an individual to solve the issue of misdiagnosis. In turn, hospitals will be able to use this model to better provide a clearer picture of one's heart condition and rightly issue and prescribe its respective treatment when needed. This would enable patients to receive help much earlier and dwindle death rates.
With rigorous tuning of the models, we deduce the best K values which best fit the models that were built. This will be expounded further on in the upcoming sections. GridSearchCV and RandomizedSearchCV were both used to optimise the various machine learning algorithms. We then analysed the results and compared our final models to our initial models using various visualisation methods.

2. DATA PREPARATION & ANALYSIS

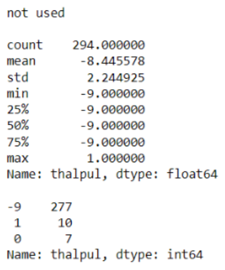
Our initial data file, hungarian.data, contained 294 instances (76 attributes) of various medical readings from cardiovascular consultations in the years ranging from 1983 to 1987 in Hungary. However, the formatting in this file proved difficult to work with in pandas since each instance of data was written over 10 lines, so we decided to realign the data to a new text file, hungarian_aligned.txt. The code used for this can be found in the Realigning_Raw_Data.ipynb notebook.
The Pre_Processing.ipynb file was where we constructed the dataset used for our models. We began by assigning names to each of the columns, as well as providing a brief description about them (found in the original data description file: heart-disease.names). After a brief analysis, we discovered that our data was essentially entirely numerical, and that there were many redundant columns consisting solely of -9.0 - the author's chosen symbol to represent missing data. Some columns also contained private information, such as social security numbers, which the author omitted and there were also modified columns that were labelled as unused. The columns that we removed based on this criteria are as follows: *ccf*, *pncaden*, *dummy*, *lvx1*, *lvx2*, *lvx3*, *lvx4*, *lvf*, *cathef*, *junk*, *name*, *restckm*, *exerckm*, *restef*, *restwm*, *exeref*, *exerwm*, *thal*, *thalsev*, *thalpul*, *earlobe*.



```
social security number

count    294.0
mean       0.0
std        0.0
min        0.0
25%        0.0
50%        0.0
75%        0.0
max        0.0
Name: ccf, dtype: float64
```



```
sum of painloc, painexer, relrest

count    294.0
mean      -9.0
std        0.0
min       -9.0
25%       -9.0
50%       -9.0
75%       -9.0
max       -9.0
Name: pncaden, dtype: float64

-9    294
Name: pncaden, dtype: int64
```



```
not used

count    294.000000
mean      -8.445578
std        2.244925
min       -9.000000
25%       -9.000000
50%       -9.000000
75%       -9.000000
max        1.000000
Name: thalpul, dtype: float64

-9    277
 1     10
 0      7
Name: thalpul, dtype: int64
```

There were also columns that were missing data in the majority of instances which we could not interpolate to the mean, median or mode, or assume a default value to, without greatly risking invalidating our data. Some columns also had extremely low variance, such as xhypo which was composed of mostly 0s as illustrated in Fig.1. With this in mind, the columns excluded are as follows: *dm*, *famhist*, *dig*, *diuretic*, *thaltime*, *xhypo*, *slope*, *ca*, *smoke*, *cigs*, *years*.

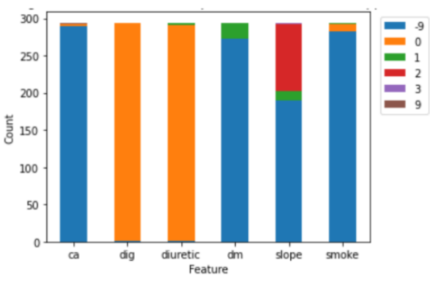
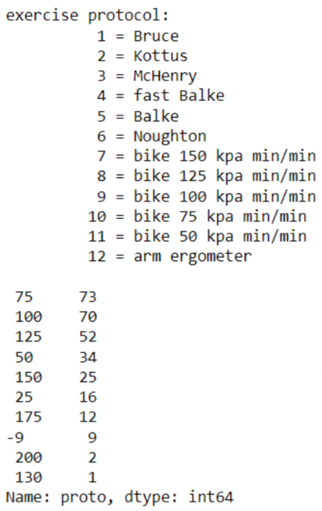


Fig.1 - Value counts of example columns that were dropped

The following attributes: *lmt*, *ladprox*, *laddist*, *diag*, *cxmain*, *ramus*, *om1*, *om2*, *rcaprox*, *rcadist*, represent major blood vessels of the heart. They are boolean fields where a value of 1 indicates that there is a greater than 50% narrowing in the respective blood vessel, and a value of 0 indicates the opposite. The author formally defined heart disease as whether any of these major blood vessels of the heart are narrowed by 50% or more, which would induce a value of 1 in the target field num and a value of 0 if all of these vessel fields are 0. We excluded these so that our models can learn from the data and generalise, rather than memorise by directly reading the answer. There were some columns which contained arbitrary numbers that would not help our models learn, such as id numbers and dates (*id*, *ekgmo*, *ekgday*, *ekgyr*, *cmo*, *cday*, *cyr*).

Fig.2 - *Proto* value counts


```
exercise protocol:
          1 = Bruce
          2 = Kottus
          3 = McHenry
          4 = fast Balke
          5 = Balke
          6 = Noughton
          7 = bike 150 kpa min/min
          8 = bike 125 kpa min/min
          9 = bike 100 kpa min/min
         10 = bike 75 kpa min/min
         11 = bike 50 kpa min/min
         12 = arm ergometer

 75     73
 100    70
 125    52
 50     34
 150    25
 25     16
 175    12
-9       9
 200     2
 130     1
Name: proto, dtype: int64
```

One column in particular, *proto*, which described the exercise protocol used for the EKG readings, contained data which did not align with the encoding used to represent each form of exercise. For instance, the author described using the numbers from 1 to 12 to represent different exercise modalities, but the most common entries range from 25 to 175 (as shown in Fig. 2); therefore, we decided to exclude this attribute from our dataset as well.

After sequentially cleaning the columns, we kept track of the rows which contained missing values to the columns we decided against dropping. We assigned random numbers between the mean (230 mg/dl) and median (237 mg/dl) of serum cholesterol values, *chol*. For the fasting blood sugar boolean field, *fbs*, we calculated the proportion of 1s present (20/286), then treated the 8 missing values as a Binomial distribution (n=8, p=20/286). We computed the expected number of 1s we would see as approximately 0.56, and so decided to assign one of these random cells a value of 1, leaving the others with a value of 0. For *htn*, *restecg*, *prop*, *nitr*, and *pro* we used the mode to interpolate this discrete data, and for *thaldur*, *met*, and *rldv5* we used either the mean or median. The instance with an id of 1053 still contained many missing values at this point, so we decided to exclude it. Before analysing any outliers, we encoded the *num* field for our binary classification task as instructed by the author, whereby if num ≥ 1, heart disease is present (represented by a 1). A value of 0 is used to indicate no heart disease (minimal risk and/or severity). We then sequentially checked for any outliers in our remaining data. We found that: *trestbps*, *chol*, *tpeakbpd*, *trestbpd* and *oldpeak* indeed had some outliers, usually above the upper fence only. However, for these attributes, we decided against replacing outliers with the mean/median/mode since these are possible values that we wanted to consider and were unlikely to be misinputs or typos. For this reason, we also avoided clipping and winsorizing. For example, we hypothesised that the higher values of blood pressure and cholesterol were likely to be key estimators of heart disease. Furthermore, there were not too many outliers and even fewer extreme outliers as shown in Fig.3.

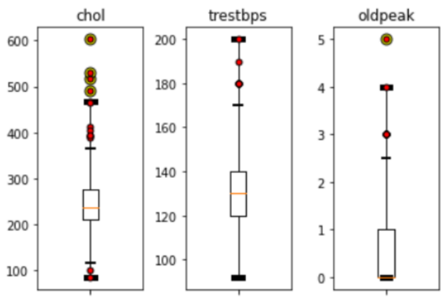
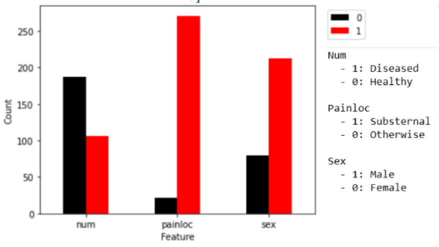


Fig.3 - Outliers (1.5 x IQR) and extreme outliers (3 x IQR)

For *thaldur*, *met*, and *thalrest*, there was only one outlier in each case so we decided that removing it would be easier and more beneficial in helping our models generalise for the average case. In the case of *rldv5* and *rldv5e*, we saw that one subject was an outlier in both fields, so we decided to remove only this instance.

Skew of columns *num*, *painloc* and *sex*



We then used min-max normalisation to scale all of our features to values between 0 and 1, thus conforming to the many boolean variables we have. This also proportionally maintains the same distance between entries, allowing us to consider our outliers as desired. Some features have a skew. For instance, there were far more subjects who reported substernal pain than other forms of pain; however, the target field had a fairly balanced distribution of data, so we decided not to up-weight or down-sample immediately. We could instead incorporate this later if it helps improve our models.
The Data_Split.ipynb file was where we split our processed data into the training, validation and test .csv files with a proportion of 0.6, 0.15 and 0.25 respectively. We ensured that we stratified on the column *num* during each split to keep the same proportion of true and false cases, as well as fixing a random state for code reusability. Our models would use the data in these files so they could be assessed fairly against one another. The final data contains 289 instances of 26 features, including the target/output, which can be viewed in the file dataML.csv.

3. LEARNING ALGORITHM SELECTION

After preprocessing our data, we constructed it for the classification category. We used a binary decision model; hence, we decided to assign Logistic Regression, kNN, and Ensemble methods to our selected dataset. By implementing these methods, the group would be in a position to predict and determine whether the patients have heart disease or not.

3.1 kNN

Since the group has done data pre-processing, it has well-structured data for the classification task, thus kNN is a suitable choice. The kNN algorithm can be used for both classification and regression, but it has certain benefits for binary decision models. To begin with, kNN tends to be simple to implement and typically provides reliable predictions, as well as having an efficient running time. Furthermore, the training process is not necessary for kNN since it is an example of instance-based learning, which means that a new data point is determined by the distance between the data points already provided. It also operates without any assumptions. kNN can be expanded to manage massive datasets; therefore, it is advantageous for our classifier when more data is provided. The results that kNN distributes ensure flexibility and coherence which may lead to an improvement in accuracy in the final result[4].

3.2 Logistic Regression

The second algorithm we decided to use is logistic regression, because it is a classification algorithm using historical data about previous outcomes, and is the most commonly used for binary decision models. The key benefit of logistic regression is that it can deal with overfitting and underfitting issues easily by implementing a shrinkage term; therefore, the accuracy of the model can be constantly raised by optimization and tuning the hyperparameters. It tends to perform especially well when data is linearly separated. In other words, it indicates significant performance for the binary decision model[5]. Furthermore, it is fairly efficient to train, and finding optimal values can be done quickly. Even though logistic regression requires a training process, whereas kNN does not, it is likely to be easier to set up the process.

3.3 Ensemble learning

There are 3 typical ensembles of learning in machine learning algorithms: bagging, boosting, and stacking. The group is going to focus more on stacking using the prior algorithms. To begin with, stacking explores the area of dissimilar models for the same task. We aim to build diverse types of learners, using them to set an intermediate prediction for each model. Afterwards, we add a new model that learns from the intermediate predictions of the matching target. The predictions of those processes for building a stack allow the model to discover nonlinear interactions that other models are not able to capture. This feature contributes to stacks having higher performance than other classification models. Moreover, stacking models typically show fewer overfitting problems than others, and provide more sturdy prediction values compared to other simple classification models[6]. Therefore, we hypothesised that we could harness the advantages offered by these varying algorithms to come up with a model that performs better than any one of them on their own via the ensemble's paradigm of "wisdom of crowds".

4. MODEL TRAINING & EVALUATION

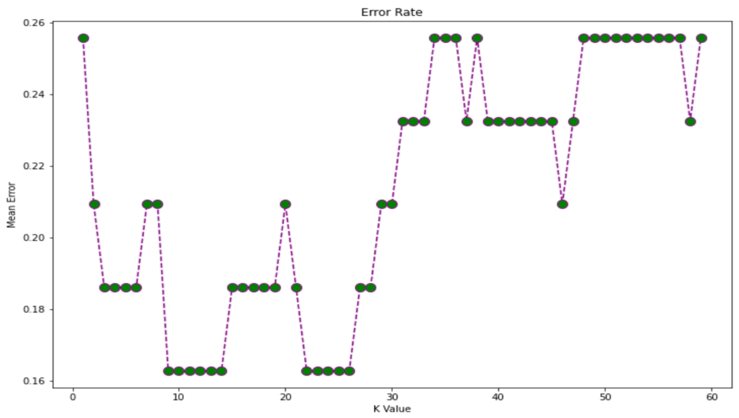
4.1 kNN

We used K-NearestNeighbors (kNN) to build one of our final models. The kNN algorithm uses 'feature similarity' to predict the values of new data points. This particular set of new data points will then be labelled accordingly to how close it is compared to the training dataset. kNN algorithms decide a number 'k', which represents the 'k' number of the nearest points to that data point that is to be classified. For example, taking the default value of 5, it will look for the 5 nearest data points.

```
[[25  2]
 [ 6 10]]
```

```
              precision    recall  f1-score   support

           0       0.81      0.93      0.86        27
           1       0.83      0.62      0.71        16

    accuracy                           0.81        43
   macro avg       0.82      0.78      0.79        43
weighted avg       0.82      0.81      0.81        43
```

After importing the test data, training data and the validation data, we made an initial model with default settings and hyperparameters, which in this case sets k to 5 with the Euclidean metric. We then fit the training data and analysed the confusion matrix of the initial model when predicting on the validation data. Precision of a model describes how much of the data are actually relevant, whereas the recall illustrates how much relevant data are being detected. We decided to use the f1 score alongside accuracy since the f1 score enables us to be more aware of the false positives and negatives that are present. F1 Score = 2*(Recall * Precision) / (Recall + Precision). Support represents the number of instances of a class in a particular dataset. This can be used to identify imbalances which may result in a structural weakness in the classifier; however, we already stratified our target (*num*) in the data preprocessing, alleviating this issue. We also concluded to use both the macro and weighted averages.



To further reinforce our accuracy and strengthen our model, we effectively tuned our hyperparameters for the 'k' value. We first used the 3-way holdout method by using a for-loop to measure the errors of various k-values in the predictions on the validation data. This was also effective at suppressing our chances of overfitting since we allowed ourselves to optimise our hyperparameters on unseen data - the validation set - separate from the training and test set. The graph above perspicuously illustrates this for 'k' values between 1 - 60. Based on the graph, it can be seen that 'k' values of 9 - 14 and 22 - 26 give the lowest mean errors. To further improve our model, we considered cross-validation techniques such as RandomizedSearchCV and GridSearchCV.

Among the two methods of RandomizedSearchCV and GridSearchCV, we selected GridSearchCV as it performs a more thorough search for the optimum hyperparameters. RandomizedSearchCV would be used later on during the ensemble stack against GridSearchCV. Overall for this particular model, GridSearchCV was shown to have outperformed any other methods despite its computational time as we will see later on. GridSearchCV also makes use of all the hyperparameters and proceeds to attempt all the various possible combinations that are passed in the dictionary. It calculates accuracy using cross-validation. Henceforth, to keep it as efficient as possible, we used the tuned hyperparameters from this method. We used the range of 9 to 26, with the increment of 2, based on the previous mean error 'k' value graph depicted earlier to provide the best possible fitting of the data. Using this range would also enable us to have a clearer picture on what to look out for while building our ensemble stack model later on. We selected 6 metrics to deal with our data to ensure a thorough assessment, namely: euclidean - the default, manhattan, minkowski, chebychev, cosine, and lastly, correlation. Euclidean distance is balanced and handles all dimensions equally. It is also very sensitive when dealing with extremities. Minkowski is the generalisation of both the euclidean distance and manhattan distance. Correlation metrics were added to find any relationship between the various variables in our model. Cosine metrics measure and find relationships between the vector spaces in our model. The leaf size hyperparameter uses a range of 1 – 40, with an increment of 5, to ensure results were still accurate whilst minimising the time taken to run the algorithm.

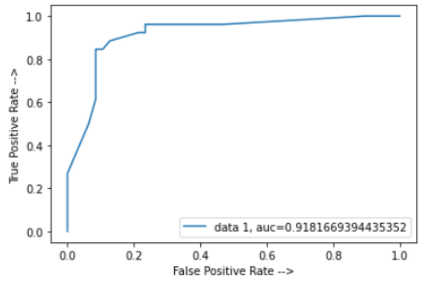
```
[[26  1]
 [ 6 10]]
```

```
              precision    recall  f1-score   support

           0       0.81      0.96      0.88        27
           1       0.91      0.62      0.74        16

    accuracy                           0.84        43
   macro avg       0.86      0.79      0.81        43
weighted avg       0.85      0.84      0.83        43
```

We then used the best hyperparameters from the results of the GridSearchCV - using accuracy for scoring - and compared the performance to that of the initial model on the validation data. By comparing the classification report and confusion matrix of this model (as shown to the right) to that of the initial model, we observed that the accuracy improved from 0.81 to 0.84. The f1 scores of this model have improved as well.

```
{'leaf_size': 1, 'metric': 'manhattan', 'n_neighbors': 15, 'weights': 'uniform'}
```

As such we used the hyperparameters as illustrated above in our final model. Lastly, to finalise the model, the training dataset and validation dataset had to be merged. We did that by concatenating the data and fitting it with the squeezed Y merged data. The final model is then tested on the test dataset, with its respective classification report and confusion matrix. Based on the final illustrated result, we received an accuracy of 0.88, with an AUC of 0.92 (rounded off) as shown in the ROC graph to the right.



```
[[42  5]
 [ 4 22]]
```

```
              precision    recall  f1-score   support

           0       0.91      0.89      0.90        47
           1       0.81      0.85      0.83        26

    accuracy                           0.88        73
   macro avg       0.86      0.87      0.87        73
weighted avg       0.88      0.88      0.88        73
```

4.2 Logistic Regression

The dataset that we chose involved classification with boolean inputs and outputs. Hence, we chose logistic regression to build our second model. The parameters used in the logistic regression are also referred to as coefficients in the model. These parameters can heavily influence the results of the final model and the predictors. The optimization algorithm can diminish the errors that occur between the observed values and predicted values by iteratively controlling the coefficients. It is completed by calculating the gradient of the cost function at each iteration and adjusting the coefficients.

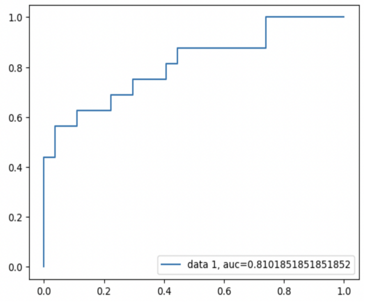
The main concern with training logistic regression models is that the overfitting and underfitting of data can occur. Overfitting occurs when the model is too complicated and starts to capture patterns in the training data that do not generalise to new data. In contrast, underfitting occurs when the model is too simple and fails to capture the underlying patterns in the data. To minimise the issues of overfitting or underfitting, it is key to use regularisation techniques, such as L1 and L2 regularisation, to introduce a bias which can improve the model's performance on the test set. Convergence issues might occur when optimization algorithms are being implemented. There may be cases in which the algorithm is not able to correctly optimise the hyperparameters to reduce errors. However, such an issue can be resolved by using a different optimization algorithm or by altering the learning rate and using different hyperparameters.

For the logistic regression modelling process, we imported and loaded the split datasets. Subsequently, we created the initial model and displayed its classification report and confusion matrix in preparation for hyperparameter tuning. This would also assist in observing any issues with the overfitting and underfitting of the data.

```
[25,  2]
[ 7,  9]
```

```
              precision    recall  f1-score   support

           0       0.78      0.93      0.85        27
           1       0.82      0.56      0.67        16

    accuracy                           0.79        43
   macro avg       0.80      0.74      0.76        43
weighted avg       0.79      0.79      0.78        43
```

The analysis of the default logistic regression model's performance on the validation set demonstrated an accuracy of 79% and an ROC of approximately 0.81. It also indicated 0.78 precision, 0.93 recall, and 0.85 f1-score for the '0' class while the other class illustrated 0.82 precision, 0.56 recall, and 0.67 f1-score. Afterwards, in order to obtain the most optimal hyperparameters for the logistic regression model, we implemented GridSearchCV.



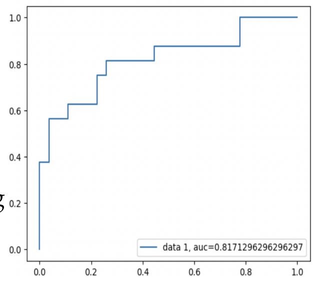
```
Best Penalty: l2
Best C: 0.05963623316594643

LogisticRegression(C=0.05963623316594643)
```

The image to the left showed the best hyperparameters after GridSearchCV where we searched penalty values between *l1* and *l2*, and 50 *C* values between 1e-4 to 10,000. We then used the 3 way hold-out method for the hyperparameter *solver* to find the best solving algorithm for these values of *penalty* and *C* on the validation set. However, it turned out that the default solver *lbfgs* outperformed the other solvers (*saga, liblinear, newton-cg, newton-cholesky, sag*). The evaluation of these hyperparameters on the validation set is shown to the right.

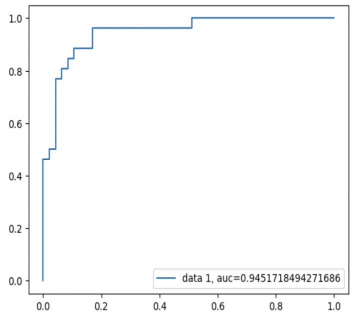
```
[26,  1]
[ 7,  9]
```

```
              precision    recall  f1-score   support

           0       0.79      0.96      0.87        27
           1       0.90      0.56      0.69        16

    accuracy                           0.81        43
   macro avg       0.84      0.76      0.78        43
weighted avg       0.83      0.81      0.80        43
```

A comparison was made between the default model and the tuned model, with the tuned model showing definite improvement. In detail, the precision for '0' was improved from 0.78 to 0.79 while the precision for '1' raised from 0.82 to 0.90. As for the recall, it illustrates that '0' sees an increase from 0.93 to 0.96, whilst '1' is maintained at 0.56. In the f1-score section, both results demonstrated progress, elevating from 0.85 to 0.87 and 0.67 to 0.69 respectively. Furthermore, accuracy was additionally improved by 2%, from 79% to 81%. The area under the curve (AUC) was slightly raised from 0.8101 to 0.8171(rounded off).



Since the result of GridSearchCV provided that the best penalty for the model is *l2*, it is going to be assigned to our final model; in other words, *l2* regularisation will be implemented for the best performance of the final logistic regression model.
The figures on the top right of the page indicate the final model's performance on the test set when trained on a combination of the training and validation sets. Each of the 3 figures showed that our data had been improved. Moreover, it continuously tends to perform worse for '1' compared to '0'.



```
              precision    recall  f1-score   support

           0       0.90      0.91      0.91        47
           1       0.84      0.81      0.82        26

    accuracy                           0.88        73
   macro avg       0.87      0.86      0.86        73
weighted avg       0.88      0.88      0.88        73
```

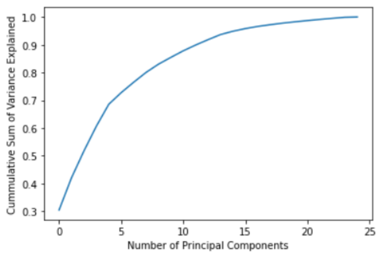
```
[43,  4]
[ 5, 21]
```

To summarise, optimization algorithms for the logistic regression task contribute to the improvement of the final model. Moreover, they provide the best values of hyperparameters which make the task much easier and more efficient.

4.3 Ensemble Stack

The final model we created was an ensemble stack classifier, which used the kNN and logistic regression algorithms as its base learners, as well as a random forest. By increasing the number of base learners in an ensemble algorithm, we are increasing our ability to closely capture the trend in our data, since these diverse algorithms highlight different trends and patterns within the data. As such, our stack was more likely to suffer from overfitting rather than underfitting due to this increase in complexity.
After importing the training, validation and testing sets into our coding environment, we trained an initial model with scikit-learn, leaving the hyperparameters at their default settings in each of the base learners. The default random forest base learner generates 100 decision trees randomly and uses the gini criterion to determine the quality of each branch in a tree, ensuring that each split results in an optimal information gain. It terminates branching when this is no longer the case. This algorithm is also a bagging ensemble itself as it then aggregates the prediction of each of these decision trees and selects the majority class as the final output for this base learner. This is passed down the pipeline into the meta learner, together with the outputs of the KNN and logistic regression.

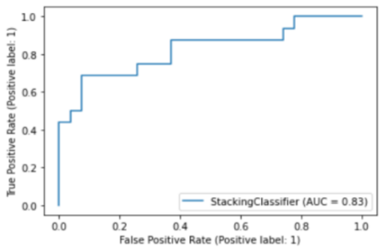
The meta-learner algorithm we used was a logistic regression as this is what is commonly used for classification tasks. It takes the predicted outputs of each of the base learners as its features (0 or 1 for our binary classification task), optimises the parameters via gradient descent and maps each instance to a value between 0 and 1 on a sigmoid function. This represents the probability of it being in one of two classes. We decided to leave the decision threshold t as 0.5, i.e. picking the most probable class, and we believe this meta-learner will perform strongly as outliers do not affect the sigmoid function as much as it affects linear estimators.
We also implemented the default 5-fold cross validation (stratified on *num*) in the StackingClassifier object to ensure we developed a robust and balanced model. This technique of cross-validation also reduced the chance of overfitting our model to the training data, especially since we do not have a vast number of instances (173 in the training set). It does this by training it on different combinations of the folds repeatedly to minimise noise from one sample alone. The default settings for our final estimator used L2-Regularisation (sum of squares) which also helped to prevent our stack from overfitting.

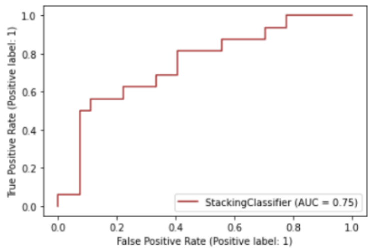


Since our training data had 26 features and only 173 instances, we decided to try a dimensionality reduction algorithm, namely principal component analysis, to see if it would improve the performance on the validation set when compared to our initial model (before hyperparameter tuning). We hypothesised that our stack may overfit to the training data because of the curse of dimensionality, whereby our data being more sparse makes it more difficult for our model to generalise. This would occur if we do not have enough instances to cover the several combinations possible between the features, causing our models to capture too much noise since the data is not dense enough to capture a trend. However, this issue of overfitting is likely minimised by the fact that many of our features are boolean, thus reducing the exponential increase in combinations. The figure above shows that 95% of the variance in our training data is explained by around 15 to 17 principal components, but after training the default model on the transformed training set and predicting values on the validation set, we observed that it performed worse. By analysing the confusion matrix below, we found that the number of true negatives and false positives worsened drastically (recall), even though there was a slight improvement in the number of true positives and false negatives. Aside from this improvement in sensitivity, all of the other measures of performance saw a statistically significant decrease, including the AUC, accuracy and precision, which are arguably the most important measures for this task. Moving forward, we decided against implementing a dimensionality reduction algorithm to further pre-process our data for all of our models.

```
Confusion Matrix:
 [[25  2]
 [ 8  8]]

              precision    recall  f1-score   support

           0       0.76      0.93      0.83        27
           1       0.80      0.50      0.62        16

    accuracy                           0.77        43
   macro avg       0.78      0.71      0.72        43
weighted avg       0.77      0.77      0.75        43
```

```
Confusion Matrix:
 [[21  6]
 [ 7  9]]

              precision    recall  f1-score   support

           0       0.75      0.78      0.76        27
           1       0.60      0.56      0.58        16

    accuracy                           0.70        43
   macro avg       0.68      0.67      0.67        43
weighted avg       0.69      0.70      0.70        43
```



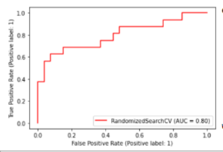


To further improve our model's performance and reduce the risk of overfitting, we began hyperparameter tuning via cross-validation techniques. Since there were 3 base learners in this model, each having their own set of hyperparameters that could be tuned, the search space for the optimization problem of finding the best hyperparameters was very large (as seen in Fig. 4). Three approaches to tuning were used.

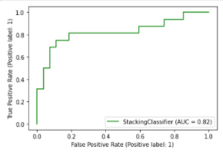
```
parameters = {
29x "kNN__n_neighbors": range(1,30),
2x  "kNN__weights": ["uniform", "distance"],
4x  "kNN__algorithm":["auto","ball_tree","kd_tree","brute"],
6x  "kNN__leaf_size":[5,10,20,30,40,50],
3x  "kNN__metric": ["euclidean", "manhattan","minkowski"],
2   "kNN__p":[1,2],
= 8352
4x  "logistic__penalty":["l1", "l2", "elasticnet", "none"],
5x  "logistic__C":[100, 10, 1.0, 0.1, 0.01],
5   "logistic__solver":["newton-cg","lbfgs", "liblinear",
= 100                       "sag", "saga"],

6x  "forest__n_estimators":[10,50,100,250,500,1000],
3x  "forest__criterion": ["gini", "entropy", "log_loss"],
2x  "forest__max_features":["sqrt", "log2"],
2   "forest__bootstrap": [True, False]
}  = 72,     So a total of 8352 x 100 x 72 = 60.1 million combinations
```

Fig.4 - Hyperparameter search space considered

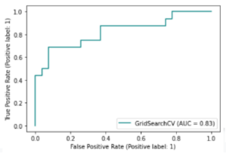
The first approach to tuning our model was to use RandomizedSearchCV to manually iterate through predictions and evaluations on the validation set, keeping track of the best hyperparameters we could find. This method combines RandomizedSearchCV with the 3-way holdout method, since we assess the performance of the random hyperparameter combinations with the best score, at each varying seed, on the validation set. Whilst this was not as reliable in theory, since we were relying on pseudo-random chance, it still resulted in an improvement to our initial model. The second method of isolated tuning involved using a more robust GridSearchCV for each of the base learners separately, individually tuning them using cross validation to find the best hyperparameters (similar to prior methods). We then assigned these optimal hyperparameters to the weak learners in our stack and assessed its performance on the validation set to find out that it mostly performed worse when compared to the first method. Aside from the AUC, all of the other metrics suffered a significant decrease, including more misclassifications on the validation set as evident in the confusion matrix. This is because this method is akin to a greedy algorithm, where at each step we select the locally optimum choice with no guarantee that we are converging towards the global optimum when we consider the stack algorithm pipeline[7]. For example, one algorithm may perform poorly under a certain subset of instances which another algorithm may handle well; however, by separately tuning them, we try to accommodate for these cases by reducing their effectiveness in their ideal subset of instances, thus decreasing the performance of the stack as a whole.



```
Confusion Matrix:
 [[26  1]
 [ 7  9]]

              precision    recall  f1-score   support

           0       0.79      0.96      0.87        27
           1       0.90      0.56      0.69        16

    accuracy                           0.81        43
   macro avg       0.84      0.76      0.78        43
weighted avg       0.83      0.81      0.80        43
```



```
Confusion Matrix:
 [[25  2]
 [ 7  9]]

              precision    recall  f1-score   support

           0       0.78      0.93      0.85        27
           1       0.82      0.56      0.67        16

    accuracy                           0.79        43
   macro avg       0.80      0.74      0.76        43
weighted avg       0.79      0.79      0.78        43
```

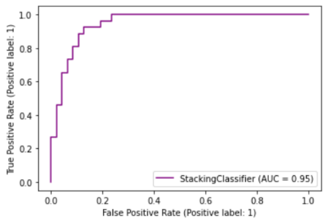


```
Confusion Matrix:
 [[25  2]
 [ 8  8]]

              precision    recall  f1-score   support

           0       0.76      0.93      0.83        27
           1       0.80      0.50      0.62        16

    accuracy                           0.77        43
   macro avg       0.78      0.71      0.72        43
weighted avg       0.77      0.77      0.75        43
```

The final method was to restrict the search space for GridSearchCV on the entire stack. The search we ended up with took 1 hour to complete and involved the most important hyperparameters present in each algorithm. For kNN, we considered the number of neighbours (*n_neighbors*, between 5, 15 & 25) used in the decision rule and the formula used to calculate the distance between instances (*metric*, as in Fig. 4). In the logistic regression, the most common *penalty* was *L2* (sum of squares rather than absolutes), so we considered this against a value of *none*, as well as different algorithms used as the *solver* (as in Fig.4). For the random forest, we considered different orders of magnitudes for the number of decision trees to generate in the bagging algorithm (*n_estimators*, between 10, 100, 1000). We also included the *criterion* which it uses to form each branch in the tree - comparing gini and log-loss with entropy. Note that in all of our random forest algorithms, we ensured the same consistent *random_state* for code repeatability. Since this method only uses cross validation to assess the scores of different hyperparameter combinations on the training set, we tested the optimal configuration on the validation set afterwards to draw comparisons with the above methods. The outcome was quite surprising since we expected this more robust method to have better performance metrics all round when compared to the prior two methods; however, it performed poorly in all of the metrics used (except AUC). This was likely a limitation of using GridSearchCV since it computed scores by using cross validation and did not incorporate the validation data like the 3-way holdout method. Hence, its scores varied and were not as reliable on the validation data when we compared predictions between these differently tuned models.
Despite the computational cost in time and memory, and performing the worst in most aspects on the validation set, we selected the third method's hyperparameters for our final model. Though seemingly counterintuitive, this method had searched through a larger search space, making it more reliable. The images below show how the third method actually had a better cross validation score of accuracy when compared to the first method (accessed via GridSearchCV.best_score_), which we already believed to be superior to the second method of isolated tuning as discussed.

```
Method 1: Best Score:  0.8040336134453782
Method 3: Best Score:  0.8268907563025211
```

```
Confusion Matrix:
 [[42  5]
 [ 3 23]]

              precision    recall  f1-score   support

           0       0.93      0.89      0.91        47
           1       0.82      0.88      0.85        26

    accuracy                           0.89        73
   macro avg       0.88      0.89      0.88        73
weighted avg       0.89      0.89      0.89        73
```



Ultimately, we trained this model on a concatenation of the training set and validation set before we tested its performance on the test set. The results indicated an 89% accuracy, as well as an AUC score of 0.95, with an appropriately optimal ROC curve indicating minimal misclassifications. The score for precision was 88% and the final parameters were as follows:

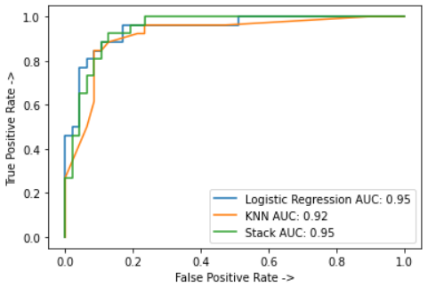
```
{'forest__criterion': 'gini', 'forest__n_estimators': 100,
 'forest__random_state': 21, 'kNN__metric': 'euclidean',
 'kNN__n_neighbors': 5, 'logistic__penalty': 'l2',
 'logistic__solver': 'newton-cg'}
```

5. MODEL COMPARISON

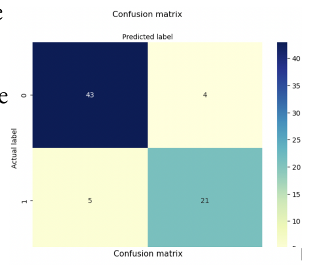
Evaluation Metric	Logistic Regression	KNN	Ensemble Stack
Accuracy	0.88	0.88	0.89
Precision	0.87	0.86	0.88
Recall	0.86	0.87	0.89
F1 Score	0.86	0.87	0.88
AUC	0.95	0.92	0.95

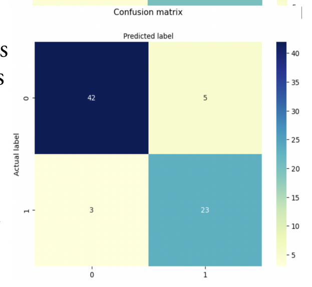
The table on the left summarises key performance metrics for each of the final models created when tested on the test set. In general, we observed similar results across the models, especially between the kNN and the logistic regression, both sharing the same accuracy. Although the logistic regression was more precise, the kNN model had higher recall and a higher f1-score; however, this could be due to the fact that the kNN misclassified a positive case as negative, thus arbitrarily improving the recall score. Nevertheless, the regression had a major

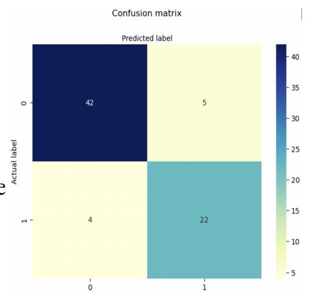
improvement in AUC, indicating that this model was expected to misclassify less. Both models were outperformed by the ensemble, which sees the highest scores all round in each of the parameters considered above, despite sharing the same value of AUC as the logistic regression.
Although the differences between the models were not very large - roughly 1-2% on average, the ensemble seemed to be the better classifier to use for our purposes of predicting heart disease. The ROC curve of the stack and the logistic regression are very similar, as depicted below, but towards the bottom left region, the logistic regression seems to be more optimal. This is because it has a higher true positive rate on average for the same rate of false positives when compared to the stack. The effect holds up until the point where they intersect at (0.1,0.9), after which the ensemble seems to dominate for the rest of the parameterisation. This likely balances out the initially optimal logistic regression curve, leading to them sharing the same AUC value of 0.95. Both curves are also more ideal than that of the kNN algorithm. This is because they are closer to filling out the perfect classifier square: 1 for true positive rate and 0 for false positive rate.



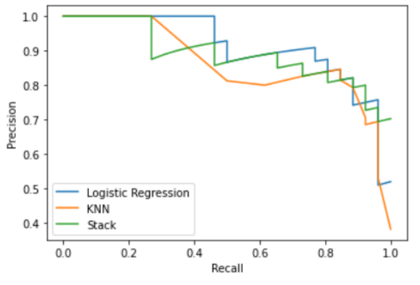
The figures to the right chromatically illustrate the confusion matrices of the final models with a heatmap. The ensemble (middle) has the highest number of true positive classifications on the test set, as well as the lowest number of false negatives. This was optimal for our classifier since it increased its sensitivity (true positives divided by the sum of true positives with false negatives). We considered this to be desirable because we would rather our model misclassify a negative case as positive than a positive case (heart disease present) as negative since this could delay the patient from receiving the treatment they require, potentially harming their life.
The kNN (bottom) and the ensemble shared the same number of true negatives and false positives, but the logistic regression (top) classified one instance better on the test set, thus improving its specificity (true negatives divided by the sum of true negatives with false positives) over the other models.







The figure below shows the ratio of precision to recall for each of the 3 models. The most optimal curve would form a square with the point (1,1) to maximise precision and recall at 1 respectively. Both the stack and the kNN models plummeted in precision much sooner than the logistic regression, with the kNN model operating at a lower precision for a more sustained duration of parameterisation before it sees an improvement. This is different to the more frequent oscillations that can be seen in the logistic regression and the stack, perhaps indicating that the logistic regression is better suited to capturing the trend in our data as it is more versatile. Another interesting observation was that the curve representing the stack closely resembled that of the logistic regression model, the main difference being that the stack suffered a decrease in precision towards the top of the graph where the regression does so towards the bottom. This indicated how the meta learner is combining the performances of these two algorithms and attempting to misclassify the least on average.



Ultimately, the results indicated similar performances between the 3 models we created, but the ensemble had a slight edge over the other two models.

6. CONCLUSION & DISCUSSION

In this project, we aimed to create a classifier capable of accurately predicting if a patient is suffering from heart disease using Hungarian heart disease data. We explored the effectiveness of simple machine learning algorithms and compared it with an ensemble method, stacking these heterogeneous algorithms to see if we could improve them.
Furthermore, the data we were using to determine our output is relatively easy for cardiologists to obtain and does not require expensive medical procedures (just an exercise EKG and some blood work), thus making this an efficient and intuitive software that can easily be implemented.
We hope that our models can be further developed into a simple program that cardiologists can use to provide a more rapid treatment to patients by streamlining their workflow.
Nevertheless, there were some drawbacks to the methods we used. When exploring the possibility of incorporating Principal Component Analysis (PCA) to improve our models, we neglected to z-score normalise (standardise) our data. This algorithm would not perform very well if the features were not appropriately scaled in this way, and we decided to use min-max normalisation in our data set instead to avoid skewing with the many boolean features we had. Perhaps it would have been beneficial if we originally standardised and appropriately allocated weights to each feature to ensure no bias was introduced in our algorithms. In fact, there are certain features that may even hinder the performance of our model since they may not play any scientific role in causing heart diseases, in which case it would have been better to remove them as inputs in the training process. Our knowledge in the domain of cardiology is limited and there is a possibility that our models suffer from this issue. The image below displays the coefficients for the logistic regression model, which can be used as evidence to argue that *restecg* and *painloc* may even be excluded since they have a near 0 coefficient. We can even use this figure to potentially draw attention to *exang* as a key indicator to heart disease, but these are merely interesting takeaways from the parameters of one algorithm and need to be further researched.

```
[[ 0.11056984  0.31584598  0.0028166   0.54522556  0.42288717  0.31438113
   0.02766424  0.10865665  0.10239813  0.15100108 -0.00264381  0.03431273
   0.18335247  0.13007608 -0.11820996 -0.07324746 -0.15637041 -0.08782699
   0.07445723  0.02209566 -0.0028343   0.68922778  0.30010712  0.06850126
   0.12314866]] [-1.85434772]
```

One of the major limitations in our project is that the data we used is considerably old (1983-1987) and there has been a major improvement in technology since it was acquired. Our data has a scarce number of instances so our models lack credibility in whether they have generalised appropriately for use in the real world. Since our model's performance is heavily dictated by the data we used, obtaining more information throughout the years up until the present day would definitely help solve this issue.
Other aspects we could improve would be the hyperparameter tuning process used. For example, we could try and run a more exhaustive GridSearchCV on a supercomputer and obtain the optimum hyperparameters for our algorithms. This would be particularly useful in the case of the ensemble since it suffers the most from an increasingly large search space.
Some key takeaways from this project would be that stacking simple algorithms together seemed to have resulted in a more capable classifier. Before we made a start on our project, we hypothesised that the ensemble stack might show the best performance for our data. After we implemented each of the 3 models, we realised that whilst the stack did indeed provide the best result, it did not perform significantly better than the simple algorithms. In fact, when considering the complexity behind creating and tuning the ensemble, it could be argued that perhaps a logistic regression may be the preferable choice.

7. REFERENCES

[1] Moy, Ernest, Barrett, Marguerite, Coffey, Rosanna, Hines, Anika L. and Newman-Toker, David E.., *Missed diagnoses of acute myocardial infarction in the emergency department: variation by patient and facility characteristics*, vol. 2, no. 1, 2015, pp. 29-40.
[2] R.Walensky, Centre for disease control and prevention, *Heart disease facts*, https://www.cdc.gov/heartdisease/facts.htm. 2022.
[3] Wong CW, Tafuro J, Azam Z, Satchithananda D, Duckett S, Barker D, Patwala A, Ahmed FZ, Mallen C, Kwok CS. *Misdiagnosis of Heart Failure: A Systematic Review of the Literature*. J Card Fail. 2021 Sep;27(9):925-933.
[4] A. Joby, *What are K-NearestNeighbors? An ML algorithm to classify data*, https://learn.g2.com/k-nearest-neighbor. 2021.
[5] A. Thanda, *What is logistic regression? A beginner's guide*, https://careerfoundry.com/en/blog/data-analytics/what-is-logistic-regression/#:~:text=Disadvantages%20of%20logistic%20regression&text=This%20is%20because%20the%20scale,the%20predictor%20(independent)%20variables. 2022.
[6] Y. Lim, *Stacked ensembles - improving model performance on a higher level*, https://towardsdatascience.com/stacked-ensembles-improving-model-performance-on-a-higher-level-99ffc4ea5523. 2022.
[7] M. Sarafanov , *Hyperparameter tuning for machine learning ensembles*, https://towardsdatascience.com/hyperparameters-tuning-for-machine-learning-model-ensembles-8051782b538b. 2022.