Heart Attack Prediction

Authors: João Reis, 98474 (50%) and Ricardo Rodriguez, 98388 (50%)
Mestrado em Engenharia Informática - DETI
Fundamentos de Aprendizagem Automática
Professor: Pétia Georgieva

***Abstract*—The purpose of this project is to implement machine learning models capable of predicting if a person is going to have an heart attack or not, according to different medical conditions. All the algorithms are compared with each other by their accuracy and we present some improvements to similar works.**

***Index Terms*—Machine Learning, Cross-validation, Hyper-parameter Tuning, Logistic Regression, Support Vector Machines, Neural Network, Naive Bayes, Random Forest Classifier, Accuracy**

I. Introduction

A heart attack, medically known as myocardial infarction, is a medical emergency that occurs when the supply of blood to the heart, which brings oxygen to the heart muscle, is severely reduced or blocked. Thus, it's very important to understand the risk factors of a heart attack and to predict it. [1]

Our project aims to train and test different machine learning models to predict whether or not a person will have a heart attack, based on many collected medical factors. For this to be possible, we will use a dataset from *Kaggle* [2] to train and tests our models. The code for this project was done using *Jupyter Notebook* [3].

This topic has gotten our attention since both of us are interested on the intersection between the machine learning and medicine fields. The idea we can train machine learning models to predict diseases or medical emergencies based on health risk factors is engaging for us, and that was the reason we chose this problem.

II. State of the art

Since the dataset is popular and publicly available at *Kaggle*, there were hundreds of public notebooks with different machine learning models and accuracy results to the same problem: predict a heart attack.

At the beginning of project, we started analysing the most popular notebooks, as well as the ones with the best models results, in order to better comprehend the authors reasoning and methodologies. This allowed us to expand our knowledge on some machine learning concepts and understand how we can improve some models accuracy.

After reading the project instructions, our highest priority was to match the requested criteria: data pre-processing, data visualization, model training (data splitting, train, validate, test, k-fold cross validation) and hyper-parameter model optimization. Additionally, we wanted to implement some popular machine learning models we had previously learned in class like Logistic Regression and SVM.

Initially, we read the most popular notebook [4] on the dataset, made by Naman Manchada, and analysed the models and their accuracies, presented in the Table I.

Table I
Naman Manchada's notebook models and accuracy

Machine Learning Model	Test Accuracy Score
Support Vector Machines	0.868852
Hypertuned SVC	0.901639
Logistic Regression	0.901639
Decision Tree	0.786885
Random Forest	0.786885
Gradient Boosting Classifier (no tuning)	0.868852

Advik Maniar's notebook [5] presented 9 different models with a better overall model accuracy than Naman Manchada's [4] one, as we can see in Table II.

Table II
Advik Maniar's notebook models and accuracy

Machine Learning Model	Test Accuracy Score
XGBoost	0.9508
AdaBoost	0.9344
MLPClassifier	0.9344
Random Forest	0.918
Gradient Boosting	0.918
Logistic Regression	0.9016
SVM	90.16
KNN	88.52
Decision Tree	81.97

In this project, our main goal is to present similar, or even better, accuracy scores than other notebooks on the *Kaggle* dataset, alongside with complementary information for this classification problem such as the confusion matrix (accuracy, precision, recall, F1 score, etc.).

III. Data Analysis

In this chapter, we will analyse the data of the dataset we chose. This dataset has the following variables:

- **age** - Age of the patient
- **sex** - Sex of the patient
- **cp** - Chest pain type — 0 = Typical Angina, 1 = Atypical Angina, 2 = Non-anginal Pain, 3 = Asymptomatic
- **trtbps** - Resting blood pressure (in mm Hg)
- **chol** - Cholestoral in mg/dl fetched via BMI sensor

- **fbs** - (fasting blood sugar 0x2264 120 mg/dl) — 1 = True, 0 = False
- **estecg** - Resting electrocardiographic results — 0 = Normal, 1 = ST-T wave normality, 2 = Left ventricular hypertrophy
- **thalachh** - Maximum heart rate achieved
- **oldpeak** - Previous peak
- **slp** - Slope
- **caa** - Number of major vessels
- **thall** - Thalium Stress Test result — (0,3)
- **exng** - Exercise induced angina — 1 = Yes, 0 = No
- **output** - Target variable

There are 303 rows and 14 columns in this dataset. As we can see in Fig.1, the heart attack distribution is almost balanced, so there is no need to scale and distribute the data which would be the best methodology if the output distribution is completely unbalanced.

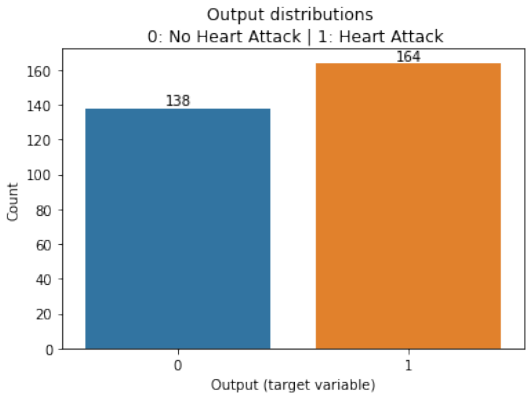

Figure 1. Heart Attack Distribution

There are 164 cases of heart attack, which represent 55% of all the dataset.

A. Correlation

Correlation matrices are fundamental to understand how our data variables are correlated with each other. It's a powerful tool to summarize the dataset, to identify and visualize patterns and what features are the most significant in the given data.

Analysing *Fig. 2*, we can identify a decent correlation between *thalachh* and *age* (0.395), *exng* and *cp* (0.393) and *slp* and *oldpeak* (0.576) features.

However, the most important thing we can induce when interpreting the correlation matrix in *Fig. 2* is the correlation between the features and the target variable *output* in order to understand which features have the most impact over the prediction of a heart attack.

In order to achieve this, we decided to group together the *output* column of the correlation matrix, sort it by descending values and plot the values in a *Fig. 3*.

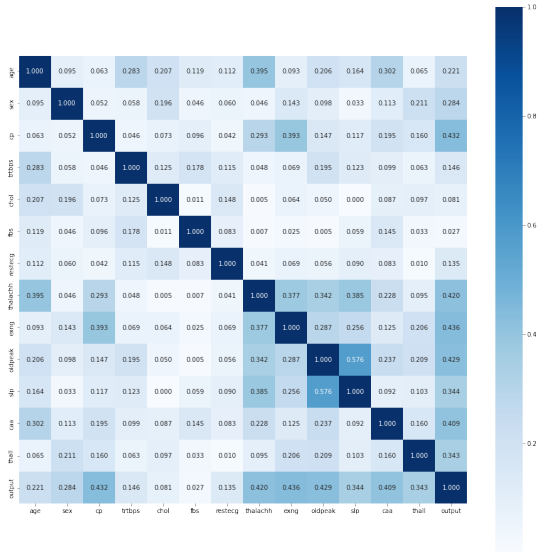

Figure 2. Correlation matrix

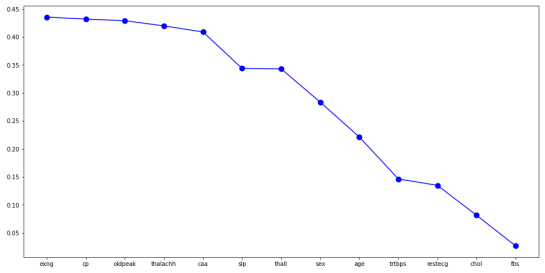

Figure 3. Correlation between features and *output* variable

Exercise induced angina (*exng*), chest pain type (*cp*), previous peak (*oldpeak*), maximum heart rate achieved (*thalacch*) and the number of major vessels (*caa*) are the features with the biggest correlation with the target variable, presenting values higher than 0.4.

B. Uni-variate Analysis

Uni-variate analysis of data is important to have a better comprehension on the dataset we are working on, providing a simple way to look at the distribution of values in a single variable.

Since the dataset has categorical (*sex, exng, caa, cp, fbs, restecg, slp, thall*) and continuous (*age, trtbps, chol, thalachh, oldpeak*) features, we decided to separate these columns and present the continuous features in *Fig. 4* and categorical features in *Fig. 5*.

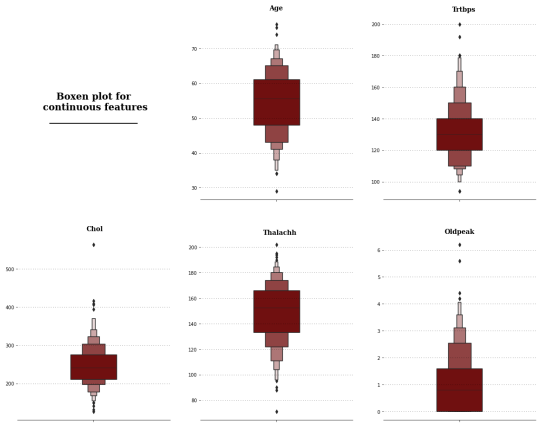

Figure 4. Boxen plot for continuous features

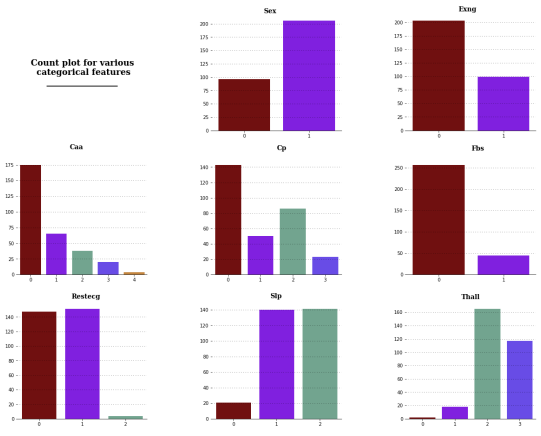

Figure 5. Count plot for categorical features

This visualization proved to be an efficient way to understand which values are the average and which values are displaced, for both categorical and continuous features.

C. Bi-variate Analysis

Bi-variate analysis is also an important think to consider when analysing data, specially when we want to find out how two different features, or variables, relate to each other in a given domain.

Although the correlation matrix displays a value representing the correlation between a feature and the target variable *output*, it fails to provide insights on how the target variable is affected for different values of the feature. Thus, *Fig. 6* demonstrates how the prediction of a heart attack changes according to continuous features, while *Fig. 7* and *Fig. 8* demonstrate the same thing but for categorical features.

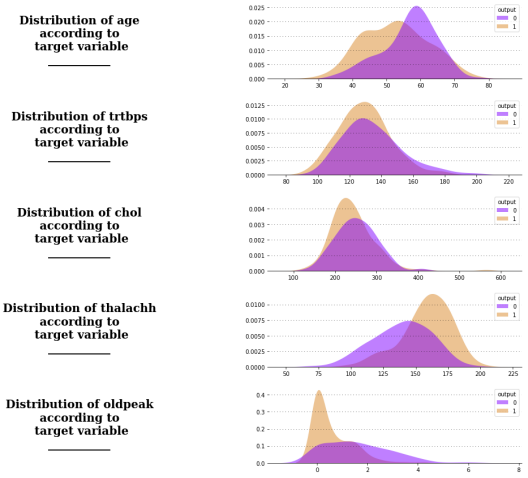

Figure 6. Bi-variate distribution of continuous features and *output*

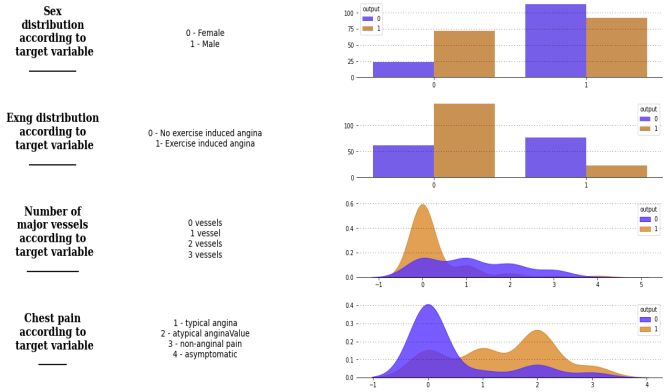

Figure 7. Bi-variate distribution of some categorical features and *output*

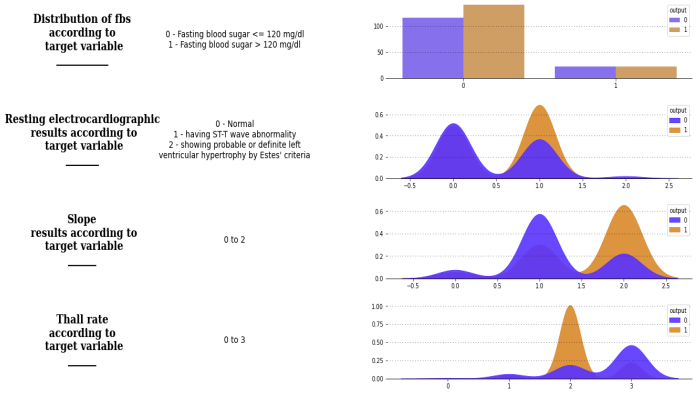

Figure 8. Bi-variate distribution of the remaining categorical features and *output*

IV. Machine Learning Models

A. Introduction

To solve this classification problem, we implemented some Machine Learning models, such as Logistic Regression, Neural Network, Gaussian Naive Bayes, Random Forest Classifier, Support Vector Machine and K-Nearest Neighbours. Initially, we tested each one of the previously mentioned models with the base/default model. Then, we did the Hyper-parameter tuning of the base model and, finally, K-Fold Cross-Validation of the hyper-tuned model was done, in order to maximize the potential of these models.

This chapter demonstrates the implementation and results of the machine learning base models, with no hyper-parameter tuning and k-fold cross validation. These last steps are presented in the next chapters (*V.* and *VI.*).

For each outcome of the models we implemented, a confusion matrix is going to be presented. The confusion matrix is used in machine learning classification problems in since it presents the number of true positives, false positives, false negatives and true negatives for the variable we're predicting. Thus, a "better" machine model is the one who has the most number of true negatives and true positives while having the smallest number of false negatives and false positives possible.

B. Logistic Regression

One of the most popular Machine Learning algorithms is Logistic Regression. This model is used for solving classification problems, like this one, and predicts the probability of a binary (1/0 or yes/no) event occurring. In this case, one (1) if the heart attack occurs, zero (0) if not.

a) ***Base Model****:* The base model confusion matrix, the classification report and the respective results are described in the following figures.

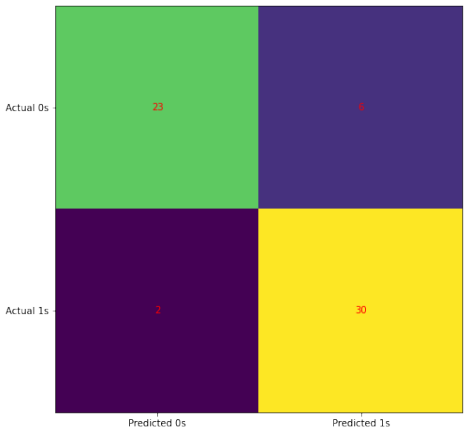

Figure 9. Logistic Regression Confusion Matrix

We are interested in the number of False Negatives, *ie*, cases when there was a heart attack but the model predicted the contrary (no heart attack). So, by analysing the last figure, we can confirm there is a low number of false negatives identified and we can consider our model as being accurate.

The classification report of Logistic Regression is presented below:

Table III
Logistic Regression Classification Report

	Precision	Recall	F1-Score	Support
0	0.92	0.79	0.85	29
1	0.83	0.94	0.88	32
accuracy			0.87	61
macro avg	0.88	0.87	0.87	61
weighted avg	0.87	0.87	0.87	61

The **accuracy** and **F1-Score** of the Logistic Regression model are both **88.52%**.

b) ***Hyper-Parameter Tuning****:* The parameters used to optimize the base model are depicted in the figure below:

Table IV
All Logistic Regression Parameters

Parameter	Values
solver	'liblinear'
max_iter	[100, 400, 800]
C	[0.1, 1, 10, 100, 1000]
class_weight	'balanced'
penalty	['l1', 'l2']

The best parameters selected for the Logistic Regression model are:

Table V
Best Logistic Regression Parameters

Parameter	Values
solver	liblinear
max_iter	100
C	10
class_weight	'balanced'
penalty	'l1'

Table VI
Hyper-tuned Logistic Regression Classification Report

	Precision	Recall	F1-Score	Support
0	0.84	0.93	0.89	29
1	0.93	0.84	0.89	32
accuracy			0.89	61
macro avg	0.89	0.89	0.89	61
weighted avg	0.89	0.89	0.89	61

The **accuracy** of the Logistic Regression model is **90.16%** while the **F1-Score** is **89.99%**.

c) ***K-Fold Cross Validation****:* For **K-Fold Cross Validation**, there was **no improvement** when applying cross validation to the hyper-tuned model. Thus, the results are exactly the same as the previous step.

C. Multi-Layer Perceptron classifier

Multi-Layer Perceptron, or MLP, is an artificial neural network consisting of three types of layers, whose nodes are called *perceptrons*. The input layer receives all the input values for the features and they are distributed to the first hidden layer of *perceptrons*. Next, the first hidden layer distributes its output to the second hidden layer of *perceptrons*, and so on. In the end, the output layer is responsible for receiving the outcome of the last hidden layer *perceptrons*. [6]

a) ***Base Model****:* For the MLP base model, we only applied one setting (*max_iter = 10000*) and the confusion matrix, as well as the classification report, is presented below:

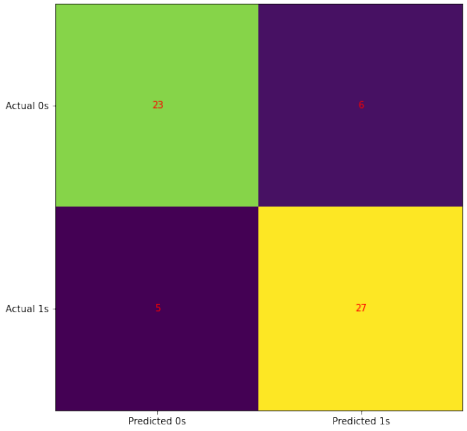

Figure 10. MLP Confusion Matrix

Table VII
MLP CLASSIFICATION REPORT

	Precision	Recall	F1-Score	Support
0	0.82	0.79	0.81	29
1	0.82	0.84	0.83	32
accuracy			0.82	61
macro avg	0.82	0.82	0.82	61
weighted avg	0.89	0.89	0.89	61

For the base model, the **Accuracy** was **81.9%** and the **F1-Score** was **83.07%**.

b) ***Hyper-Parameter Tuning****:* The parameters used to optimize the base model are depicted in the figure below:

Table VIII
ALL MLP PARAMETERS

Parameter	Values
hidden_layer_sizes	[(2,),(3,),(4,),(6,),(8,)]
activation	['logistic', 'tanh']
solver	['adam']
max_iter	[2000, 10000]

There's no perfect formula to select the correct number of hidden layers and nodes in a neural network. However, there are several studies providing different thumb rules such as the approximation proposed by Masters (1993). [7]

$$n = \sqrt{i * o} \tag{1}$$

The number of nodes in a hidden layer formula is the result of the square root of the multiplication between the number of input (i) and output (o) neurons. This equation helped us to select the set of *'hidden_layer_sizes'* values.

Table IX
BEST MULTI-LAYER PERCEPTRON PARAMETERS

Parameter	Values
hidden_layer_sizes	(3,)
activation	'logistic'
solver	'adam'
max_iter	10000

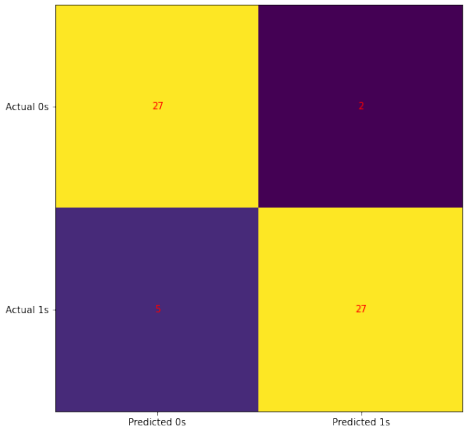

Figure 11. Hyper-tuned MLP Confusion Matrix

Table X
HYPER-TUNED MLP CLASSIFICATION REPORT

	Precision	Recall	F1-Score	Support
0	0.84	0.93	0.89	29
1	0.93	0.84	0.89	32
accuracy			0.89	61
macro avg	0.89	0.89	0.89	61
weighted avg	0.89	0.89	0.89	61

For the hyper-tuned model, the **Accuracy** was **88.52%** and the **F1-Score** was **88.52%**, a big leap in comparison to the base model results.

c) ***K-Fold Cross Validation****:* The Accuracy and F1-Scores for the MLP hyper-tuned model after K-Fold Cross Validation is the same as if there was no cross validation. The confusion matrix, as well as the classification report, is presented below:

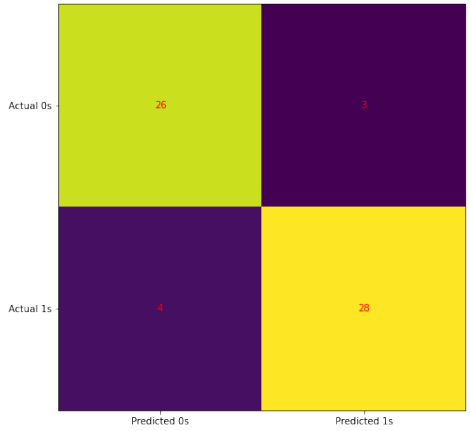

Figure 12. K-Fold Cross Validated MLP Confusion Matrix

Table XI
K-FOLD CROSS VALIDATED MLP CLASSIFICATION REPORT

	Precision	Recall	F1-Score	Support
0	0.87	0.90	0.88	29
1	0.90	0.88	0.89	32
accuracy			0.89	61
macro avg	0.88	0.89	0.89	61
weighted avg	0.89	0.89	0.89	61

D. Gaussian Naive Bayes

Gaussian Naive Bayes, or GNB, is a probabilistic machine learning algorithm based on the Bayes theorem [8] heavily used for classification problems. [9].

*a) **Base Model**:* The confusion matrix and classification report for the Gaussian NB base model is presented below:

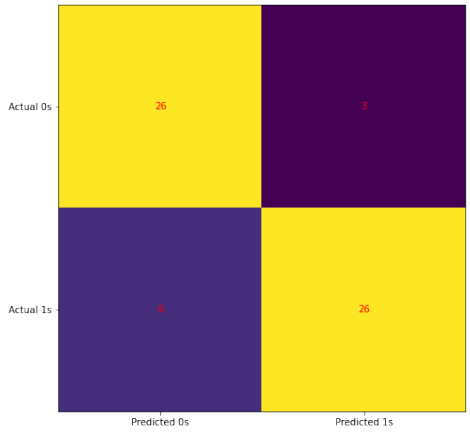

Figure 13. Gaussian NB Confusion Matrix

Table XII
GAUSSIAN NB CLASSIFICATION REPORT

	Precision	Recall	F1-Score	Support
0	0.81	0.90	0.85	29
1	0.90	0.81	0.85	32
accuracy			0.85	61
macro avg	0.85	0.85	0.85	61
weighted avg	0.86	0.85	0.85	61

For the base model, the **Accuracy** and **F1-Score** was **85.24%**.

*b) **Hyper-Parameter Tuning**:* The parameters used to optimize the base model are depicted in the figure below:

Table XIII
ALL GAUSSIAN NB PARAMETERS

Parameter	Values
var_smoothing	np.logspace(0,-9, num=100)

Table XIV
BEST GAUSSIAN NB PARAMETERS

Parameter	Values
var_smoothing	1.0

The hyper-tuned Gaussian NB Confusion Matrix, as well as the Classification Report, has **exactly the same values** as the base model results.

*c) **K-Fold Cross Validation**:* The Accuracy and F1-Scores for the Gaussian NB hyper-tuned model after K-Fold Cross Validation is the same as if there was no cross validation, since it has the **same values as the previous step**.

E. Random Forest Classifier

In Random Forest Classifier, or RFC, there is a large number of individual trees. Each one of these trees presents its outcome in ensemble and the most common output between all of them becomes the prediction for the model. [10]

*a) **Base Model**:* For the MLP base model, we only applied one setting (*max_iter = 10000*) and the confusion matrix, as well as the classification report, is presented below:

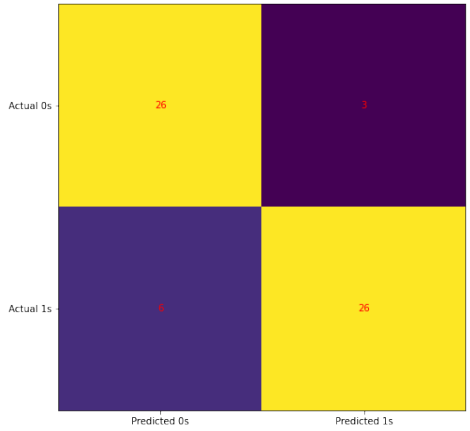

Figure 14. RFC Confusion Matrix

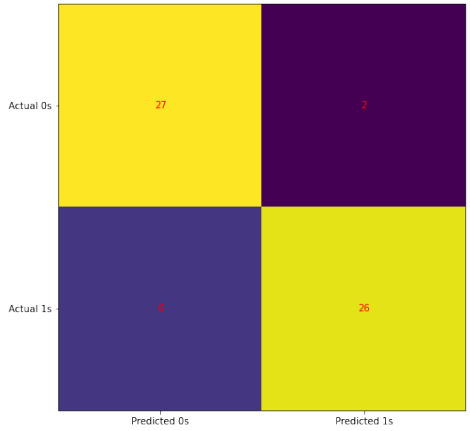

Figure 15. Hyper-tuned RFC Confusion Matrix

Table XV
RFC CLASSIFICATION REPORT

	Precision	Recall	F1-Score	Support
0	0.81	0.90	0.85	29
1	0.90	0.81	0.85	32
accuracy			0.85	61
macro avg	0.85	0.85	0.85	61
weighted avg	0.86	0.85	0.85	61

For the base model, the **Accuracy** and **F1-Score** was **85.24%**.

*b) **Hyper-Parameter Tuning**:* The parameters used to optimize the base model are depicted in the figure below:

Table XVI
ALL RFC PARAMETERS

Parameter	Values
max_features	['auto', 'sqrt']
criterion	['gini', 'entropy']

Table XVII
BEST RFC PARAMETERS

Parameter	Values
max_features	'auto'
criterion	'entropy'

Table XVIII
HYPER-TUNED RFC CLASSIFICATION REPORT

	Precision	Recall	F1-Score	Support
0	0.82	0.93	0.87	29
1	0.93	0.81	0.87	32
accuracy			0.87	61
macro avg	0.87	0.87	0.87	61
weighted avg	0.88	0.87	0.87	61

For the hyper-tuned model, the **Accuracy** was **86.88%** and the **F1-Score** was **86.66%**, a very small improvement in comparison to the base model results.

*c) **K-Fold Cross Validation**:* The **Accuracy** and **F1-Scores** for the RFC hyper-tuned model after K-Fold Cross Validation **are the same as if there was no cross validation**.

F. Support Vector Machine

Support Vector Machine (SVM) is one of the most robust prediction method. This model consists on create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. The extreme vectors or support vectors define the gap between categories.

*a) **Base Model**:* In the base model, we use the model as explained above without any optimization. The confusion matrix, as well as the classification report, is presented below:

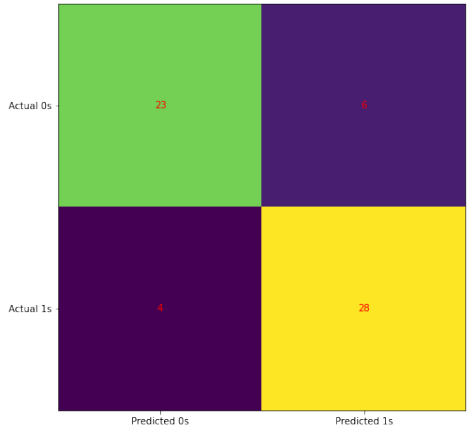

Figure 16. SVM Confusion Matrix

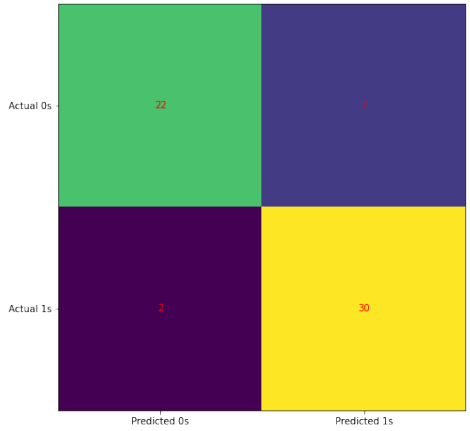

Figure 17. Hyper-tuned SVM Confusion Matrix

Table XIX
SVM CLASSIFICATION REPORT

	Precision	Recall	F1-Score	Support
0	0.85	0.79	0.82	29
1	0.82	0.88	0.85	32
accuracy			0.84	61
macro avg	0.84	0.83	0.83	61
weighted avg	0.84	0.84	0.84	61

The **accuracy** obtained with this model was **83.6%** but the **F1 Score** was a little bit higher with **84.8%**.

*b) **Hyper-Parameter Tuning**:* The parameters used to optimize the base model are depicted in the figure below:

Table XX
ALL SVM PARAMETERS

Parameter	Values
C	[1, 2, 3, 4, 5, 6, 7, 8, 9]
gamma	[0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5]

Table XXI
BEST SVM PARAMETERS

Parameter	Values
C	8
gamma	0.005

Table XXII
HYPER-TUNED SVM CLASSIFICATION REPORT

	Precision	Recall	F1-Score	Support
0	0.92	0.76	0.83	29
1	0.81	0.94	0.87	32
accuracy			0.85	61
macro avg	0.86	0.85	0.85	61
weighted avg	0.86	0.85	0.85	61

The accuracy obtained with this the hyper-parameters tuning was better than the accuracy on base model. The percentage value of **accuracy** is **85.2%** but the **F1 Score** is **87.0%**.

*c) **K-Fold Cross Validation**:* All the results using K-Fold Cross Validation, ie, accuracy and F1 Score percentage, confusion matrix and classification report, are equal to the results obtained using **Hyper-Parameter Tuning**.

G. K Nearest Neighbours

K-Nearest Neighbour is one of the simplest Machine Learning algorithms. This model assumes the similarity between the new data and available cases and put the new data in the category that is most similar to the available categories. K-NN model can be used for Regression as well as for Classification but mostly it is used for the Classification problems, like this one.

*a) **Base Model**:* In the base model, we use the model as explained above without any optimization. We only applied one setting, (*n_neighbors = 1*). The confusion matrix, as well as the classification report, is presented below:

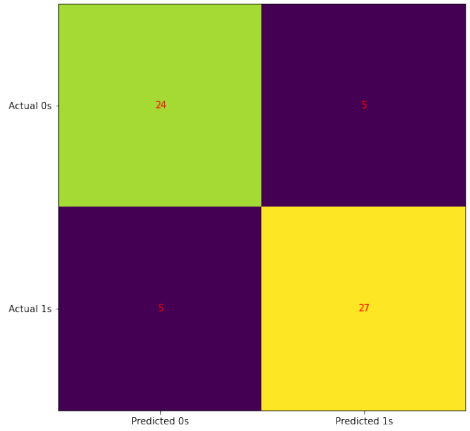

Figure 18. K-NN Confusion Matrix

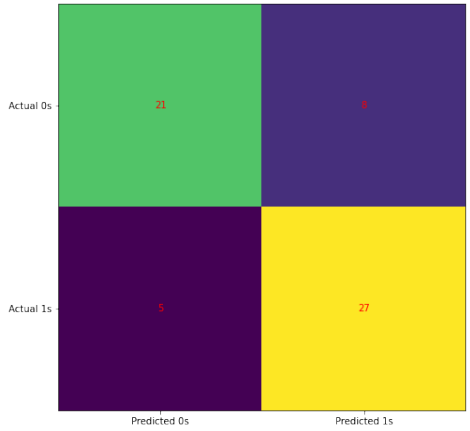

Figure 19. Hyper-tuned K-NN Confusion Matrix

Table XXIII
K-NN CLASSIFICATION REPORT

	Precision	Recall	F1-Score	Support
0	0.83	0.83	0.83	29
1	0.84	0.84	0.84	32
accuracy			0.84	61
macro avg	0.84	0.84	0.84	61
weighted avg	0.84	0.84	0.84	61
6				

The **accuracy** obtained with this model was **83.6%** but the **F1 Score** was a little bit higher, with **84.4%**.

*b) **Hyper-Parameter Tuning**:* The parameters used to optimize the base model are depicted in the figure below:

Table XXIV
ALL K-NN PARAMETERS

Parameter	Values
leaf_size	[1, 2, ..., 50]
n_neighbors	[1, 2, ..., 30]
p	[1, 2]

Table XXV
BEST K-NN PARAMETERS

Parameter	Values
leaf_size	1
n_neighbors	21
p	1

The confusion matrix, as well as the classification report, is presented below:

Table XXVI
HYPER-TUNED K-NN CLASSIFICATION REPORT

	Precision	Recall	F1-Score	Support
0	0.81	0.72	0.76	29
1	0.77	0.84	0.81	32
accuracy			0.79	61
macro avg	0.79	0.78	0.78	61
weighted avg	0.79	0.79	0.79	61

The number of False Negatives, ie, cases when there was a heart attack but the model predicted the contrary (no heart attack), increased by three units. This is bad, because the goal is try to decrease the number of False Negatives.

The accuracy obtained with this the hyper-parameters tuning was, surprisingly, worse than the accuracy on base model. The percentage value of **accuracy** is **78.7%** but the **F1 Score** is **80.6%**.

*c) **K-Fold Cross Validation**:* All the results using K-Fold Cross Validation, ie, accuracy and F1 Score percentage, confusion matrix and classification report, are equal to the results obtained using **Hyper-Parameter Tuning**.

V. MODEL COMPARISON

The figure below presents a box plot with information related to the accuracy of all models used in this project.

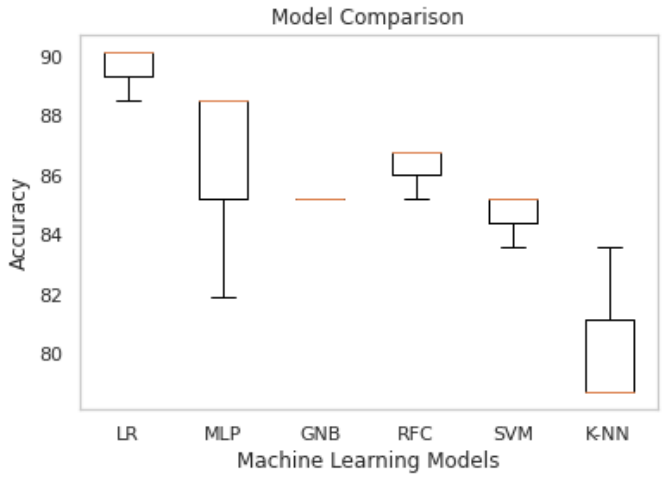

Figure 20. Model Accuracy Comparison

By analysing the figure above, we conclude that the **best model** for this problem is **Logistic Regression**, because it got the highest accuracy, whereas the **K-NN model** has the **worse accuracy**.

VI. Novelty and Contributions

In comparison to Naman Manchada's notebook, we both have the same results for the Logistic Regression model and is the best algorithm for the problem. Additionally, our Random Forest Classifier model had a 86.88% of accuracy while Manchada's one had 78.68%. However, his hyper-tuned SVC model had a bigger accuracy (90.16%) than ours (83.60%).

When analysing Advik Maniar's notebook, we noted that he uses a specific *random_state* value (65) for his models, which generates significantly better accuracy results (90%) in our models, as we tested out. However, it was very interesting seeing how some models we never heard of, like XGBoost and AdaBoost, which had amazing results.

VII. Conclusion

This was our first machine learning project and, in retrospective, we consolidated many important concepts learned in class. We've learned the importance of preparing the data, testing different parameter settings to improve the base model, as well as cross validating the dataset, and deepen our knowledge on several machine learning algorithms.

Overall, the obtained results were fairly positive, considering that the dataset only had 303 rows for training and testing all the different machine learning models. A bigger dataset would be better to improve the accuracy of our models, since 13 features is a decent number and they require more data to be efficiently trained.

We conclude that each model has different accuracies and confusion matrices. Thus, it's important to think and to use different approaches and see the algorithms with the best results.

We're looking forward to learn more about machine learning and to implement new solutions to important problems like this one.

References

[1] "What is a heart attack?" (2022) www.heart.org. Available at: https://www.heart.org/en/health-topics/heart-attack/about-heart-attacks.

[2] Rahman, R. (2021) "Heart attack analysis & prediction dataset", Kaggle. Available at: https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset.

[3] João Reis & Ricardo Rodriguez "Heart Attack Prediction", GitHub. Available at: https://github.com/joaoreis16/heart_attack_prediction

[4] Namanmanchanda (2021) "Heart attack - eda + prediction (90% accuracy)", Kaggle. Kaggle. Available at: https://www.kaggle.com/code/namanmanchanda/heart-attack-eda-prediction-90-accuracy/notebook-container.

[5] Advikmaniar (2021) "Heart Attack-EDA/Prediction with 9 model(¿95%)", Kaggle. Kaggle. Available at: https://www.kaggle.com/code/advikmaniar/heart-attack-eda-prediction-with-9-model-95/Compare-Accuracy-and-Execution-Time.

[6] Multilayer Perceptron an overview, ScienceDirect Topics. Available at: https://www.sciencedirect.com/topics/computer-science/multilayerperceptron.

[7] Masters, T. (1993). "Practical neural network recipes in C++". Academic Press.

[8] Joyce, J. (2003) Bayes' theorem, Stanford Encyclopedia of Philosophy. Stanford University. Available at: https://plato.stanford.edu/entries/bayes-theorem/.

[9] Gaussian naive bayes: What you need to know? (202AD) upGrad blog. Available at: https://www.upgrad.com/blog/gaussian-naive-bayes/.

[10] Understanding random forest - towardsdatascience.com. Available at: https://towardsdatascience.com/understanding-random-forest-58381e0602d2.