Christian Stingone (s4644983) & Alex Di Stefano (s4638131)

Heart Disease

Heart Failure Prediction Dataset

Machine Learning course 2021/2022

Intro to the Dataset

This dataset contains data from patients hospitalized for presumed cardiovascular diseases. Inside, we can find both healthy patients and those with heart disease. Obviously, all the data relating to their state of health are present, as we will see later. We have chosen it because in the medical world prevention is never too much and through research it is possible to prevent and treat many diseases that could otherwise be fatal.

People with cardiovascular disease or who are at high cardiovascular risk need early detection and management wherein a machine learning model can be of great help.

Kaggle: https://www.kaggle.com/fedesoriano/heart-failure-prediction

What Will We Do

In this project we want to analyze this dataset with the purpose of predict the probability of a person to have an heart disease/failure, as well as interpreting affected factors on patient status. To do that we choose different ML Models and we want to compare them to find the best one for our purpose.

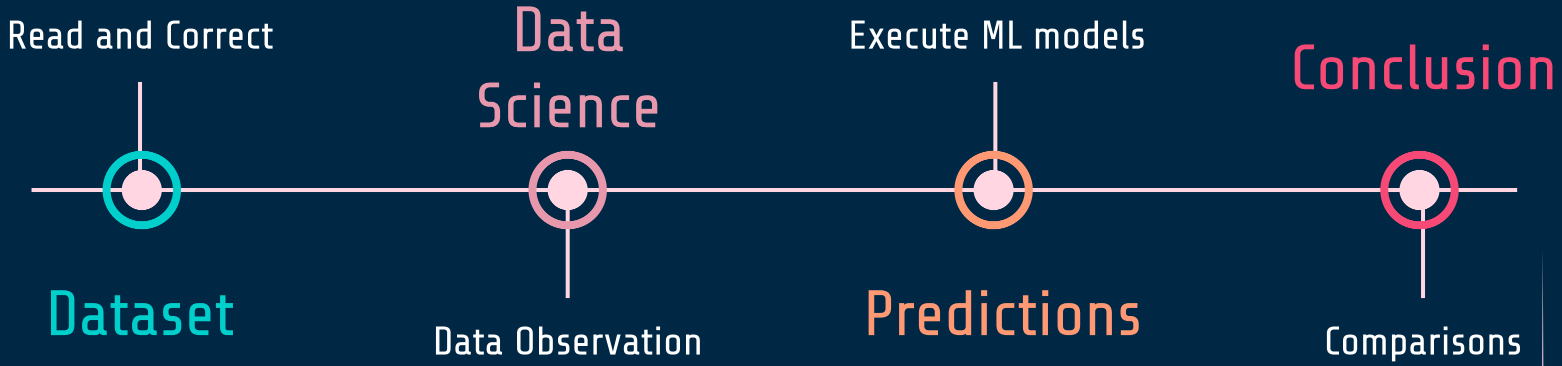
Random Forest

The simplest and most reliable

KNN

The "Must" of machine learning

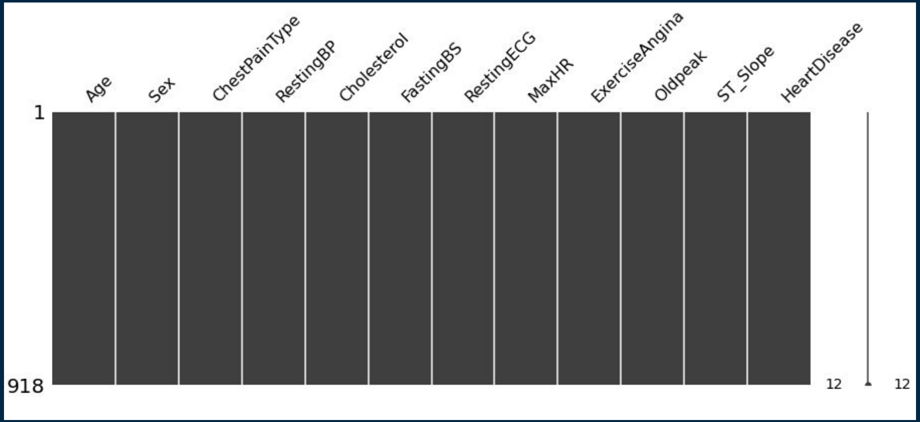
Subdivision of the Notebook



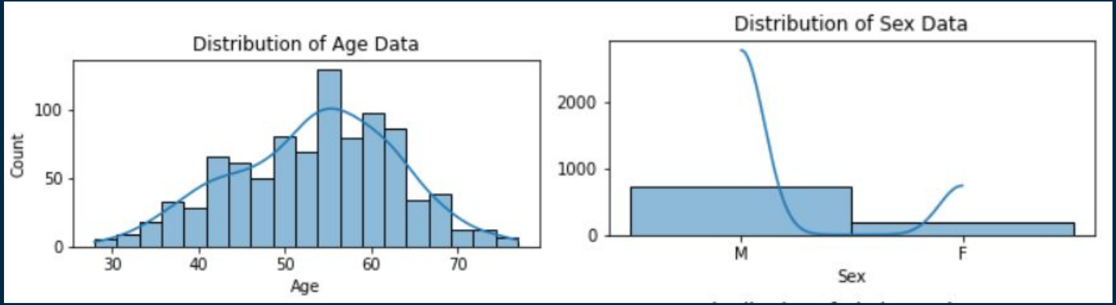
Correction of the Dataset

In this dataset we have no columns with null values and no duplicate rows.

However, we noticed that some columns have wrong values, probably badly recorded during observation on the patient. Precisely they are the columns relating to cholesterol and restingBP. To correct this error we took the incorrect values and replaced them with the average of the entire column, in order to minimize the error on those rows.

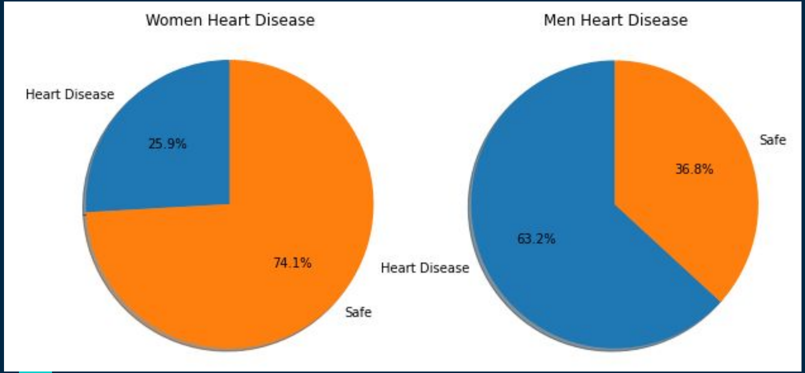


Observation on the Dataset



Composition

In these graphs we see how our dataset is mainly composed of men and the most frequent age is about 55 years



Heart disease

In these graphs we see how men suffer more from heart disease.
Or rather, in our dataset there are more men who have suffered a heart disease than women

Visit the notebook for more observations

Predictions

How Does the Classification Work?

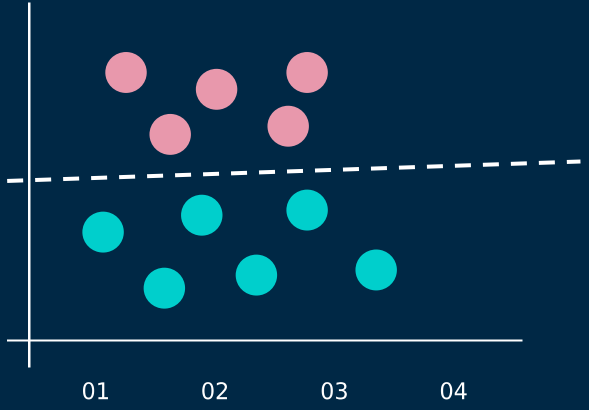
The classification algorithm is a supervised learning technique that is used to identify the category of new observations based on the training data.

The goal of the classification algorithm is to identify the category of a given data set, and these algorithms are primarily used to predict the output for categorical data.

There are two types of classifications:

- **Binary Classifier**: If the classification problem has only two possible outcomes.
- **Multiclass Classifier**: If a classification problem has more than two results.

We are using K-NN and Random Forest for a Binary Classification problem

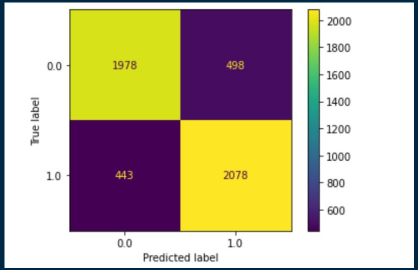


ROC and False Positive Matrix

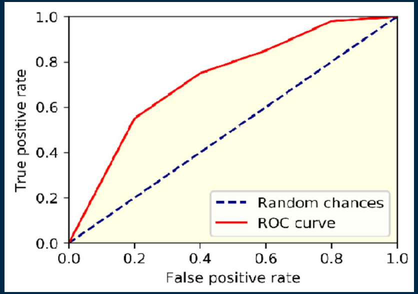
First we will train the model on the train data and then we will run it on the test data.

Precision: The ratio between correct elements and total elements will be expressed as a percentage, this will tell us the accuracy of the prediction.

False Positive Matrix: This matrix is composed of 2 rows and 2 columns and indicates the predicted values corresponding to the real values. In this way we could evaluate both which targets were predicted correctly and which ones were wrong.



ROC: This concept is plotted with the ROC curve (Receiver operating characteristic). True Positive Rate (TPR, fraction of true positives) on Y axes and False Positive Rate (FPR, fraction of false positives) on X axes. The yellow area is called AUC and tells us if we have many correct values over the wrong ones

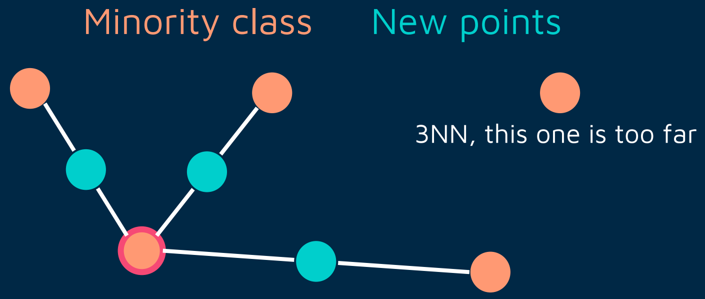


SMOTE

Before moving on to creating the ML models we must first improve the dataset so that our models run at their best. We do this by converting all values to numbers and applying the SMOTE algorithm.

When use SMOTE?
Our dataset is slightly unbalanced, the number of elements with HeartDisease at 0 has approximately 1/4 fewer elements than HeartDisease at 1 and this could affect accuracy to ours ML models. To balance our data we use SMOTE.

How does it work?
It takes the minority class and makes KNN among its points. It takes n points, draws a line between them, along the line creates a new point with the correct values to fit on that line. As many lines are drawn and as many points are created as needed to have a balanced dataset



Random Forest

Random Forest – Theory

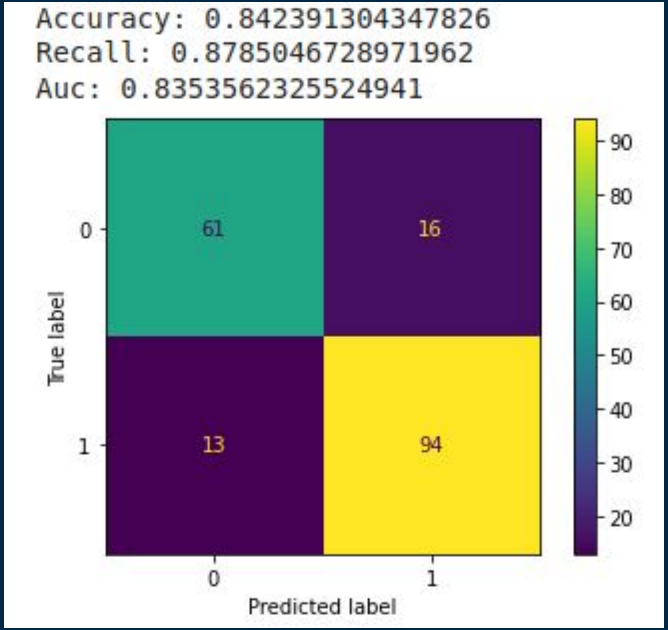
Hyperparameters:

- **Bootstrap values**: Number of values extracted from the bootstrap ~ default: 2/3
- **Max features**: number of features I look at each node ~ default: sqrt(n_features)
- **Max depth**: depth of trees / number of leaves ~ default: None

Other important parameters:

- **Number of trees** ~ default: 100
- **Algorithm for voting** ~ default: Majority Voting
- **Min sample leaf**: the min number of samples required to be at a leaf node ~ default: 1
- **Min sample split**: the min number of samples required to split an internal node ~ default: 2

Random Forest – Execution



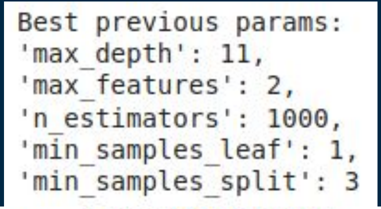
```
Accuracy: 0.842391304347826
Recall: 0.8785046728971962
Auc: 0.8353562325524941
```

Random Forest – Parameter Tuning

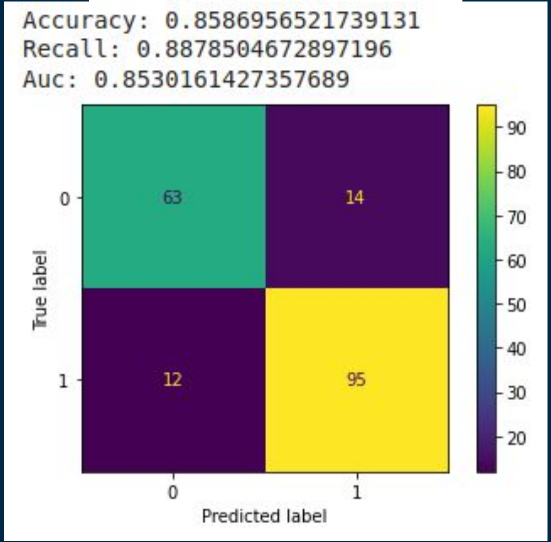
The parameter tuning needs in order to find the best parameters.

The parameters we are going to use are:

- **Max Depth**: depth of each the tree
- **Number of Estimators**: the number of trees
- **Max Features**: these are the maximum number of features Random Forest is allowed to try in individual tree.
- **Min_samples_leaf**: The minimum number of samples required to be at a leaf node.
- **Min_samples_split**: The minimum number of samples required to split an internal node



```
Best previous params:
'max_depth': 11,
'max_features': 2,
'n_estimators': 1000,
'min_samples_leaf': 1,
'min_samples_split': 3
```



```
Accuracy: 0.8586956521739131
Recall: 0.8878504672897196
Auc: 0.8530161427357689
```

Random Forest – Conclusion

The Random Forest offers good results and excellent execution speed. Furthermore already in the version with the default parameters we obtain excellent precision.
This result is so good also thanks to the SMOTE method to balance our dataset, as this predictive model works best on datasets that are not too unbalanced.

Subsequently we tried to do the tuning of the parameters although this model doesn’t require a particular tuning as the improvements that can be obtained are minor, as in our case.

So you should be wondering if it is worth running a GridSearchCV, which takes several minutes/hours to run, rather than using the model with the basic parameters.

K-NN

K-NN - Theory

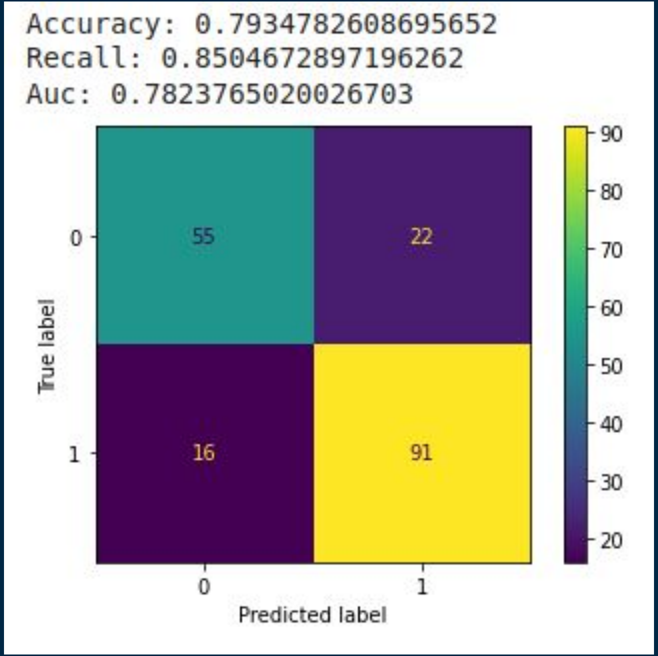
Hyperparameters:

- **K**: number of neighbors ~ default: 5

Other important parameters:

- **Metric**: distance metric formula ~ default: minkowski

K-NN - Execution


```
Accuracy: 0.7934782608695652
Recall: 0.8504672897196262
Auc: 0.7823765020026703
```

K-NN - Parameter Tuning

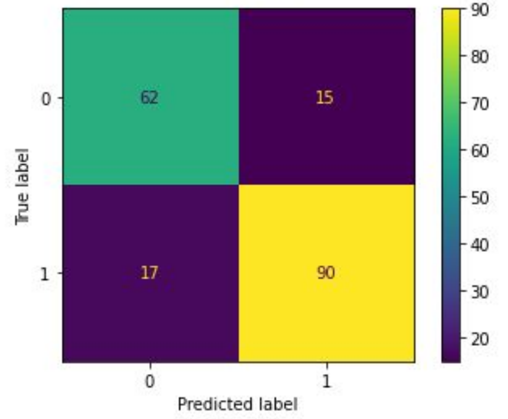
The parameter tuning needs in order to find the best parameters.

The parameters we are going to use are:

- **N_neighbors**: number of neighbors
- **Metric**: let's try various formulas to calculate the distance ("euclidean", "manhattan", "chebyshev", "minkowski")

```
Best previous params:
'n_neighbors': 9,
'metric': 'manhattan'
```

```
Accuracy: 0.8260869565217391
Recall: 0.8411214953271028
Auc: 0.823158150260954
```



K-NN - Conclusion

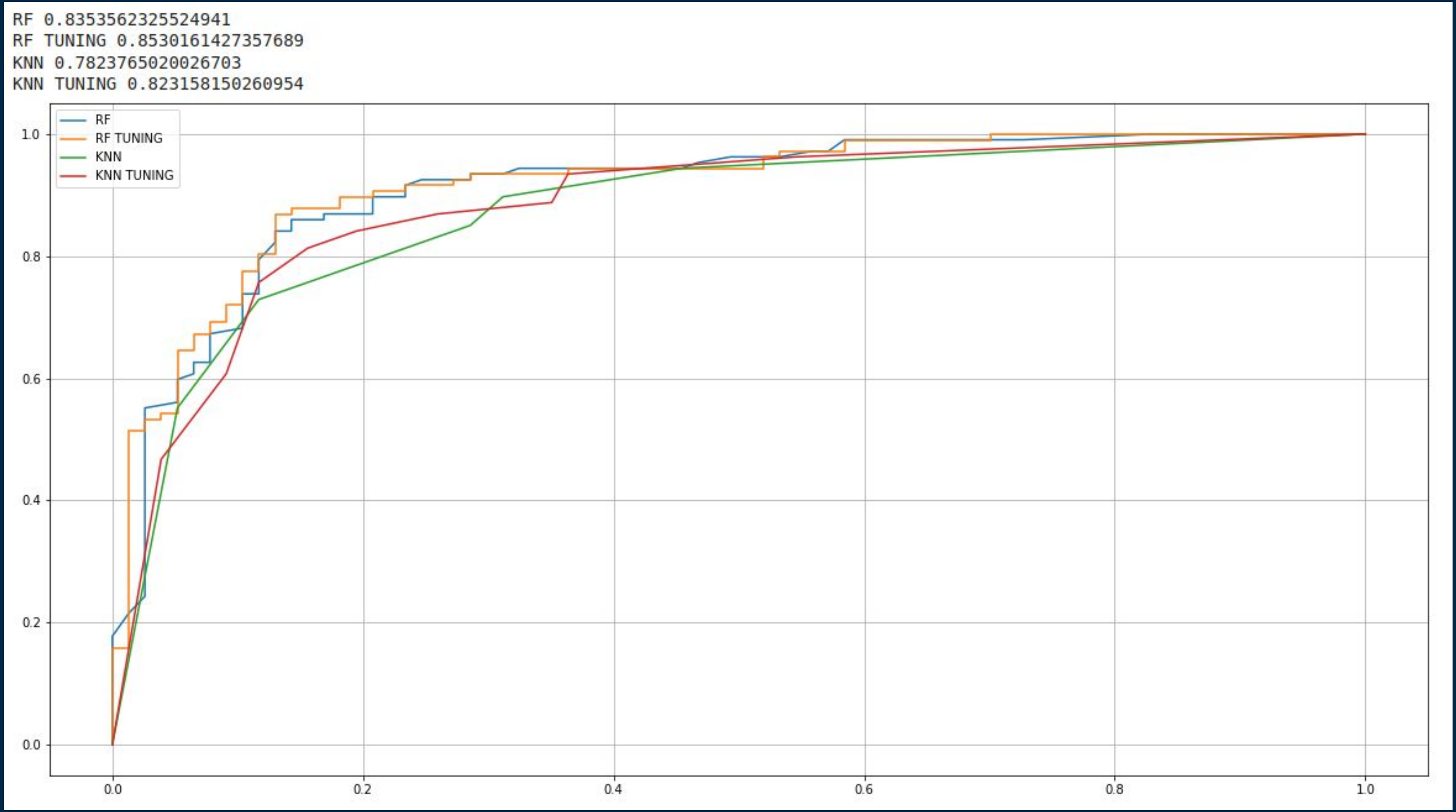
We must note that its default accuracy is less than the Random Forest, this is caused by the number of neighbors which must be chosen carefully and which can be considered its most important parameter.

In fact, in the section dedicated to the tuning of the parameters we can see how the precision increases as the number of neighbors increases. Thus obtaining excellent precision, recall and AUC values.

The only disadvantage of this algorithm is that it becomes slow if the number of examples / predictions / independent variables increases.

KNN is a lazy learner, there is no function and therefore the model is as large as the data.

Comparisons



```
RF 0.8353562325524941
RF TUNING 0.8530161427357689
KNN 0.7823765020026703
KNN TUNING 0.823158150260954
```

KNN

Accuracy: ~82%

Recall: ~85%

AUC: ~82%

Random Forest

Accuracy: ~86%

Recall: ~88%

AUC: ~85%

Conclusions

In this project we prefer the Random Forest (also without tuning the parameters), remembering however that we must first execute the SMOTE function (which uses KNN) to balance the dataset, because it has a better accuracy, recall and AUC.

These data, together, give us a good overview of the goodness of this model

How to improve: by adding more data (medical or about individual’s life) or adding another ML model, like XGBoost and/or weighted-KNN

Applications: in any medical area where you want to prevent an heart disease/attack

Do you have any questions?

THANKS

View full project at
Colab Notebook

Christian Stingone (s4644983) & Alex Di Stefano (s4638131)