Heart Disease Prediction and Classification using Machine Learning

Foundations of Artificial Intelligence - Prof: Petia Georgieva (petia@ua.pt) - Academic Year 2021/22

André Reis Fernandes
Number: 97977
Computacional Engineering
Physics Department
andre.fernandes16@ua.pt
WorkRate : 50%

Gonçalo Jorge Loureiro de Freitas
Number: 98012
Computacional Engineering
Physics Department
goncalojfreitas@ua.pt
WorkRate : 50%

***Abstract*—In this document we use 6 Machine Learning algorithms implemented both in Python and in the Rapid Miner software to predict heart disease using a given data set as well as study which model is the best.**

***Index Terms*—machine learning, artificial intelligence, confuse matrix, performance metrics, label, parameters, hyperparameters tunning, ROC, AUC, Cost/Loss function.**

I. Introduction

This project was proposed for the Foundations of Artificial Intelligence class by our teacher Petia Georgieva, an 3rd year class of the 1st cycle degree in Computational Engineering. The interest in learning and use Artificial Intelligence and Machine Learning to our day-to-day problems is growing day by day especially in the use to predict diseases, because manually determine the odds of getting, for example, an heart disease based on risk factors it's difficult. We decided to choose this theme because it reveals great importance to the society as well for us that learn more about AI and ML.

II. The data set

The dataset used in this article is the Cleveland Heart Disease dataset taken from the UCI repository.The dataset consists of 297 individuals data. There are 14 columns in the dataset, which are described below.

1. Age, in years
2. Sex, 1 = male; 0 = female
3. cp: chest pain type
   - Value 0: typical angina
   - Value 1: atypical angina
   - Value 2: non-anginal pain
   - Value 3: asymptomatic
4. trestbps: resting blood pressure (in mm Hg on admission to the hospital)
5. chol: serum cholestoral in mg/dl
6. fbs: (fasting blood sugar $>$ 120 mg/dl) 1 = true; 0 = false
7. restecg: resting electrocardiographic results
   - Value 0: normal
   - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of $>$ 0.05 mV)
   - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8. thalach: maximum heart rate achieved
9. exang: exercise induced angina, 1 = yes; 0 = no
10. oldpeak = ST depression induced by exercise relative to rest
11. slope: the slope of the peak exercise ST segment
    - Value 0: upsloping
    - Value 1: flat)
    - Value 2: downsloping
12. ca: number of major vessels (0-3) colored by flourosopy
13. thal: 0 = normal; 1 = fixed defect; 2 = reversable defect and the label
14. condition (target) : 0 = no disease, 1 = disease

III. The approach

To study this data set we implemented 6 models,using both Python the Rapid Miner software. The code, rapid miner processes and the data can be found in our github repository

In this article we will be using the following classification models for classification :

1) Logistic Regression (Python)
2) Support Vector Machine (Python)
3) Gaussian Naive Bayes (Python)
4) Decision Tree (Python)
5) K-Nearest-Neighbor (Python)
6) Neural Networks (Rapid Miner)

It is commom to use a 60:20:20 ratio which the middle 20% is the validation set which is used to find the hyper-parameters, but in this article, for all models, we divide the data in an 80:20 ratio, that is, the train set is 80% and test set is 20% of the whole data. The train set will be used to train the model, i.e, finding the parameters that minimizes our error in the training set.

A. Data Overview

Firstly we need to check if there is null values that can mislead us. The data that we use is already pre-processed, which means, there aren´t any null values, so we can jump right into applying the models. If there were null values we needed to drop/eliminate them to ensure an higher performance.

Let's take a look on how the data is arranged by comparing each feature with the frequency and condition, remembering that condition = 1 (orange) implies a disease and = 0 (green) implies that there isn't a disease

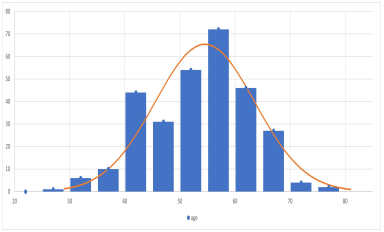

Fig. 1: Age distribution following a normal distribution.

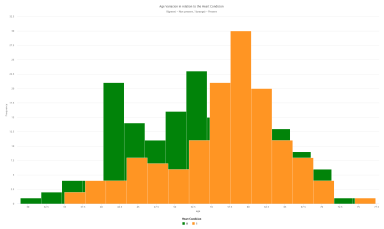

Fig. 2: Age Variation in relation to the Heart Condition.

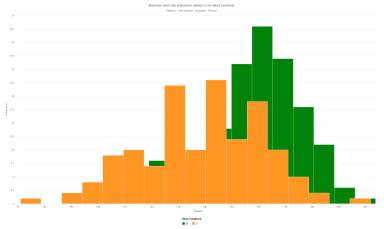

Fig. 3: Maximum heart rate achieved in relation to the Heart Condition.

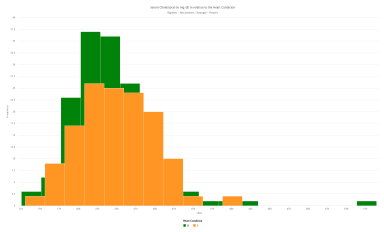

Fig. 4: Serum Cholesterol (in mg/dl) in relation to the Heart Condition.

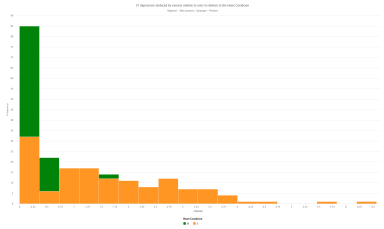

Fig. 5: ST depression (induced by exercise relative to rest) in relation to the Heart Condition.

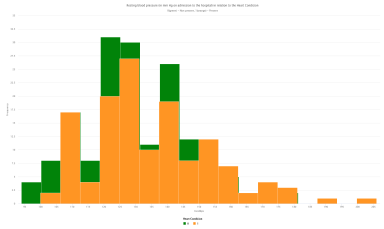

Fig. 6: Resting blood pressure (in mm Hg on admission to the hospital) in relation to the Heart Condition.

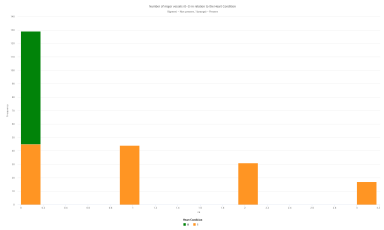

Fig. 7: Number of major vessels (0-3) in relation to the Heart Condition.

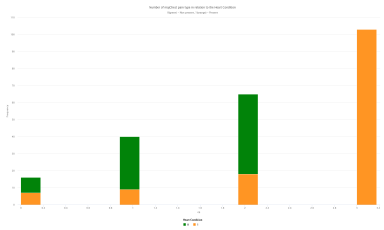

Fig. 8: Number of Chest pain type in relation to the Heart Condition.

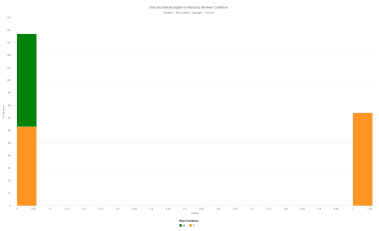

Fig. 9: Exercise induced angina in relation to the Heart Condition.

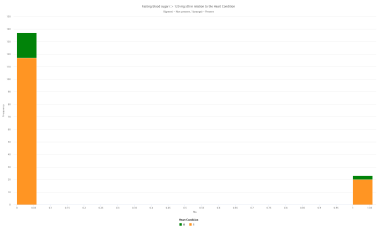

Fig. 10: Fasting blood sugar ($>$ 120 mg/dl) in relation to the Heart Condition.

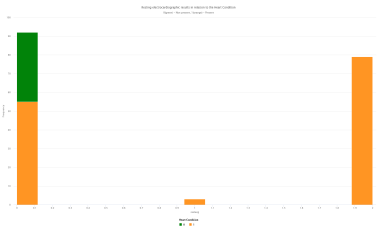

Fig. 11: Resting electrocardiography results in relation to the Heart Condition.

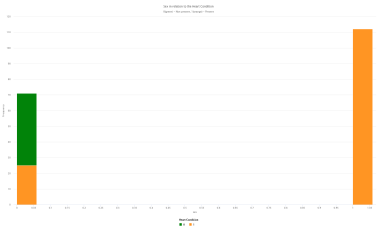

Fig. 12: Sex in relation to the Heart Condition.

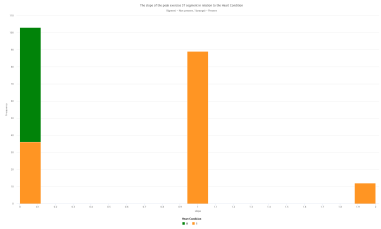

Fig. 13: The slope of the peak exercise ST segment in relation to the Heart Condition.

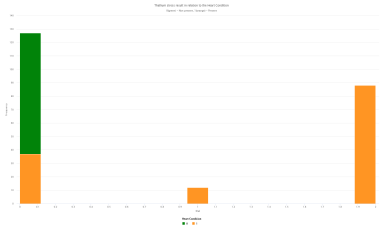

Fig. 14: Thallium stress result in relation to the Heart Condition.

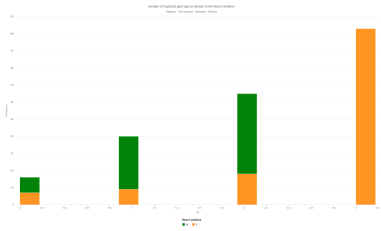

Fig. 15: Type of chest pain in relation to the Heart Condition.

	age	*sex*	*cp*	*trestbps*	*chol*	*fbs*	*restecg*	*thalach*
age	1							
sex	-0.0924	1						
cp	0.110471	0.008908	1					
trestbps	0.290476	-0.06634	-0.03698	1				
chol	0.202644	-0.19809	0.072088	0.131536	1			
fbs	0.132062	0.03885	-0.05766	0.18086	0.012708	1		
restecg	0.149917	0.033897	0.063905	0.149242	0.165046	0.068831	1	
thalach	-0.39456	-0.0605	-0.33931	-0.04911	-7.5E-05	-0.00784	-0.07229	1
exang	0.096489	0.143581	0.377525	0.066691	0.059339	-0.00089	0.081874	-0.38437
oldpeak	0.197123	0.106567	0.203244	0.191243	0.038596	0.008311	0.113726	-0.34764
slope	0.159405	0.033345	0.151079	0.121172	-0.00922	0.047819	0.135141	-0.38931
ca	0.36221	0.091925	0.235644	0.097954	0.115945	0.152086	0.129021	-0.26873
thal	0.120795	0.370556	0.266275	0.130612	0.023441	0.051038	0.013612	-0.25839

Fig. 16: Correlation Matrix

If we look now to Fig.2 we can see that as the age increases the number of persons with an heart diseases increases as well. From the Fig.3 we can see that the lower the thalach the higher probability the person with a disease. From Fig.4 and Fig.9 we conclude that there ins't a correlation between the chol and if the exercises induces angina or not with the condition. The Fig.5, Fig.6 and Fig.7 shows us that the higher the old peak, trestbps and the number of major vessels the more probability the person has to have an disease. Figures 8 and 10 shows that when the chest pain type is asymptomatic there is more cases of an condition while when studying the fasting blood sugar we see that if this is < 120 the cases raises as well. Fig.11 shows that there are more heart diseases when the resting electrocardiograph results shows probable or definite left ventricu- lar hypertrophy by Estes' criteria (Val 2). Fig.12 tell us that the males (1) are more prominent to an heart disease. From Fig.13 we see that the he flat exercise ST segment traduces in an higher probability of a condition. Finally, Fig.14 show us that the reversible defect thal leads to an higher number of conditions. With Fig.16 we can see how each feature correlates with one and another to the heart disease.

IV. Models

After a quick research we found that we can use the scikit-learn's GridSearchCV, which identifies the best parameters that you can included in the parameter grid based on pre-defined parameters by the users. This will be use to help us find the parameters that minimizes the error in the train set, by define the scoring parameter of GridSearchCV to 'neg_mean_absolute_error'. This function is going to be used for the first 5 models. In the last model we use a different approach that will be explained in chapter IV-F.

A. Logistic Regression

This type of statistical analysis (also known as logit model) is often used for predictive analytics and modeling, and extends to applications in machine learning. It is used in statistical software to understand the relationship between the dependent variable and one or more independent variables by estimating probabilities using a logistic regression equation.

In order to map predicted values to probabilities, we use the sigmoid function. The function maps any real value into another value between 0 and 1

$$\mathcal{S}(z) = \frac{1}{1 + \exp -z}$$

As this function returns a value between 0 and 1 we select a threshold value or tipping point above which we will classify values into class 1 and below which we classify values into class 0. This threshold is simply the value returned and if its ≥ 0.5 it will belong to the class 1, i.e, class positive (with disease), if lower than 0.5 the model would assume that there isn't an disease. [14]

There are many parameters for this model,for example the 'penalty' parameter; which has 4 possible parameters 'none',

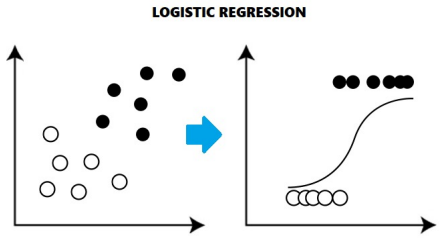

Fig. 17: Logistic Regression idea of model working

'l2' (disperse the error terms in all the weights that leads to more accurate customized final models), 'l1' (gives output in binary weights from 0 to 1 for the model's features and is adopted for decreasing the number of features in a huge dimensional dataset) and 'elasticnet' (l2 + l1 penalty); or the 'solver' parameter; with the options being 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'; and many more that can be found in the official documentation with the description of what each parameter does. Has we want the best accuracy we need to see which combination of these parameters gives us the lest error in the train set. Using the GridSearchCV function, where we compare every possible combination, between some predefined values of C, penalty and solver, it was found that the best parameters are $C : 0.1, penalty : l2, solver :' newton - cg'$, with all the other parameters are set as default. The max_iter parameter was also change to an higher value than the default value (100) to make sure that our model converges.

B. Support Vector Machine (SVM)

A Support Vector Machine is a supervised machine learning algorithm which can be used for both classification and regression problems. It follows a technique called the kernel trick to transform the data and based on these transformations, it finds an optimal boundary between the possible outputs.

The main idea is to identify the optimal separating hyper-plane which maximizes the margin of the training data. Let us understand this objective term by term. A separating hyper-plane it's a line (or other, depending on the dimension) that can separate the given data, which means, from one side we have all elements from one group and in the other side we have the elements of the another group. Maximizing the distance between the nearest points of each class and the hyper-plane would result in an optimal separating hyper-plane. This distance is called the margin.[1]

Using GridSearchCV, we found that the best possible parameters between $kernel = ['rbf','poly','sigmoid'], C = [0.1, 1, 10, 100, 1000], gamma = [1, 0.1, 0.01, 0.001, 0.0001]$ the best are $kernel =' poly'$, $C = 1$ and $gamma = 0.0001$.

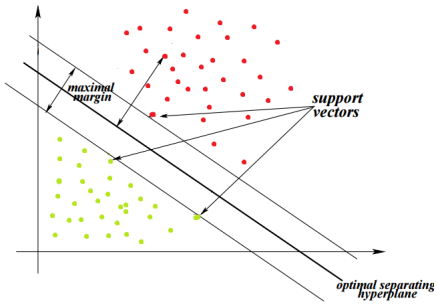

Fig. 18: Optiomal sparating hyper-plane

Outlook	*Temp*	*Humidity*	*Windy*	*Golf?*
rainy	hot	high	false	no
rainy	hot	high	true	no
overcast	hot	high	false	yes
sunny	mild	high	false	yes
sunny	cool	normal	false	yes
sunny	cool	normal	true	no
overcast	cool	normal	true	yes
rainy	mild	high	false	no
rainy	cool	normal	false	yes
sunny	mild	normal	false	yes
rainy	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
sunny	mild	high	true	no

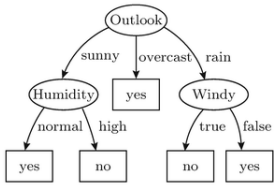

Fig. 19: Example of a Decision Tree

C. Naive Bayes (Gaussian)

Naive Bayes is the easiest and rapidest classification method available, and it is well suited for dealing with enormous amounts of information. In several applications such as spam filtering, text classification, sentiment analysis the Naive Bayes classifier has shown to be effective. It makes predictions about unknown classes using the Bayes theory of probability.

There are 3 types of Naive Bayes Classification, the Gaussian, the Bernoulli and the Multinomial. In this article we just use the first one, which is employed when the predictor values are continuous and are expected to follow a Gaussian distribution, as ours does as we can see in Fig 1. It is worthy to note that this model is the easiest to program as it doesn´t have any parameters to tunning.

1) Bayes Theorem: Bayes Theorem can be used to calculate conditional probability. Being a powerful tool in the study of probability, with the following formula.[2]

$$\mathcal{P}(A|B) = \frac{\mathcal{P}(B|A)(A)}{\mathcal{P}(B)}$$

Where $\mathcal{P}(A|B)$ is the probability of event A occurring, given event B has occurred; $\mathcal{P}(B|A)$ the probability of event B occurring, given event A has occurred;$\mathcal{P}(A)$ the probability of event A and $\mathcal{P}(B)$ the probability of event B. Note that events A and B are independent events (i.e., the probability of the outcome of event A does not depend on the probability of the outcome of event B).

D. Decision Tree

Decision Trees are powerful machine learning algorithms capable of performing regression and classification tasks. To understand a decision tree, let's look at an simple example which tries to predict if it is save to play golf, in Fig.19. We start at the root of the tree that contains our training data. At the root, we split our dataset into distinguished leaf nodes, following certain conditions like using an if/else loop. These splitting criteria are carefully calculated using a splitting technique.

With this tree we can easily decide if we can play golf in good conditions of it's better to stay home.

So all-in-all, decision trees are a hierarchical series of binary decisions, and the antecedent nodes are simply the best split for the available training data at each level of our decision tree.

As we to minimize the error we also want the best splits, and these traduce in wanting the smallest tree and the most 'purest' (most homogeneous) nodes and for this we need to understand some fundamental splitting parameters that it uses to define those conditions, like *Gini Index*, *Entropy*, *Information Gain*, and *Classification error*.[9]

1) Gini Index: Every Machine Learning model has a loss function or a cost function, whose objective is to minimize the cost, i.e., the tentative distance between the predicted value and actual value. *Gini Index* is the cost/loss function that is used by decision trees to choose which feature will be used for splitting the data, and at what point the column should be split, and can be determined by:

$$GINI(node) = 1 - \sum_{Class_j} [\mathcal{P}(Class_j|node)]^2 \qquad (1)$$

$P(Class_j|node)$ being the probability of $Class_j$ at a given node.

An lower *Gini* score traduces in a lower cost which gives us the information that it was a good split.

2) Classification Error: Suppose that we cut off the growing process at various points over the growing process, and we evaluate the error of the tree at that point and time. This would lead to a graph of size vs. error (where error is the probability of making a mistake). There are two error rates to be considered; training/test error, i.e. fraction of mistakes made on the training/test set. These error curves are as follows:

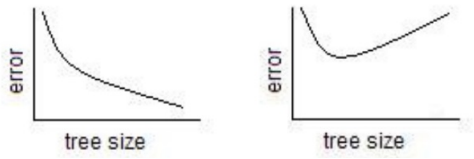

Fig. 20: Training (left) and Test error curves (right)

As the tree size increases, training error decreases. However, as the tree size increases, testing error decreases at first since we expect the test data to be similar to the training data, but

at a certain point, the training algorithm starts training to the noise in the data, becoming less accurate on the testing data. At this point we are no longer fitting the data but instead we are fitting the noise in the data. Therefore, the shape of the testing error curve will start to increase at a certain point at which the tree is too big and too complex to perform well on testing data (an application of Occam's razor). This is called overfitting to the data, in which the tree is fitted to spurious data. As the tree grows in size, it will fit the training data perfectly and not be of practical use for other data such as the testing set. We want to choose a tree at the minimum of the curve, but we are not aware of the test curve during training. We build the tree only using the training error curve, which appears to be decreasing with tree size. Again, we have two conflicting goals. There is a trade off between training error and tree size.

This error can be calculated using the follow equation:

$$Error(node) = 1 - \max_{Class_j} (\mathcal{P}(Class_j|node)) \tag{2}$$

3) Entropy: Entropy, *H*, measures the randomness or disorders in a system. In terms of data, we can define it as the randomness in the information we are processing. The higher the randomness, the higher the entropy. Hence, harder to conclude from that information, so we ideally want $H = 0$. Mathematically, we calculate entropy as:

$$H(node) = -\sum_{Class_j} [\mathcal{P}(Class_j|node) log(\mathcal{P}(Class_j|node))] \tag{3}$$

4) Information Gain: Information gain, I_{gain}, measures the amount of information provided by a given feature or attribute about a particular target class, i.e. how much uncertainty was reduced after splitting on a given feature. While creating a decision tree, our goal is to find the attribute having the highest Information Gain, and conversely, the lowest entropy. Mathematically, it is calculated as the difference of the initial and final entropy.

$$I_{gain}(node) = \mathcal{H}_i - \mathcal{H}_f \tag{4}$$

5) Working with data: While training, our decision tree model evaluates all possible splits across all possible columns and picks the split with the lowest Gini Score. With the first split, all the data according to a specific condition falls towards either the left or the right of the root node. Now, for each side of training data under the root node, all possible splits are calculated again and the split with the lowest Gini Index is chosen. The process repeats for both left and right sides till we reach the terminating nodes representing a class in the target column.

6) Finding the best Parameters: Now that we have explained the importance of the splitting parameters we need to find the best ones for our dataset. As with first 2 models explained above we can use the GridSearchCV to obtain the parameters giving us the better possible accuracy.

We also need to pay attention than if we use Information Gain as a criterion, we assume that our attributes are categorical, and as per Gini index, we assume that our attributes are continuous. For our data set, we will work with Gini Index.

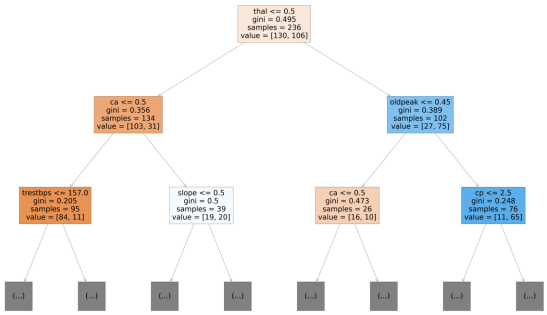

Fig. 21: Decision Tree for our data with $max_{depth} = 2$

You can create the tree to whatsoever depth using the max_{depth} parameter, only two layers of the output are shown above. Let's break the blocks in the above visualization:

1) $thal <= 0.5$ Is the condition on which the data is being split. (where thal is the column name).
2) Gini: Is the Gini Index. Although the root node has a Gini index of 0.495, which is not so great, we can imagine what the other Gini scores would have looked like.
3) Samples: The number of data rows before the split.
4) Values=[x,y]: Provides the split rows of training data into the following leaf nodes. Since we are doing binary classification, there are only two values.

Now all we have to do is find the hyper-parameters. For this model we will we use the GridSearchCV which give us back that the best values which are max_depth=5, max_features='log2', max_leaf_nodes=30, min_samples_leaf=8, min_weight_fraction_leaf=0.3. It's possible to get other parameters and results due to the randomness of this model, i.e, for example, a different first split would change everything. So this aren't the ultimate best parameters but the best parameters between some tests we made.

E. K-Nearest-Neighbor (KNN)

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. This is a very used algorithm due to being simple, easy to implement and very versatile as it can be used for classification, regression, and search problems. However it got one big disadvantaged, the algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase. [5]

This algorithm can be explained in few steps as following:

1) Select the number of the neighbors, K
2) Calculate the Euclidean (5) distance of K number of neighbors

3) Take the K nearest neighbors as per the calculated Euclidean distance
4) Among these K neighbors, count the number of the data points in each category
5) Assign the new data points to that category for which the number of the neighbor is maximum
6) The model is ready

With the Euclidean Distance of A(X_1, Y_1) and B(X_2, Y_2) being:

$$ED_{<A,B>} = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2} \quad (5)$$

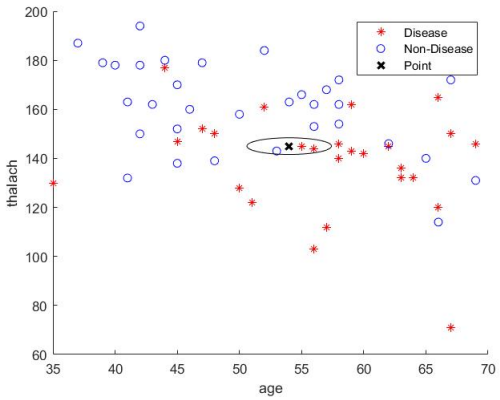

Fig. 22: Example of K = 3 for our train set of thalach-age

As we can see in the figure above, after selecting $K = 3$ and the model choosing an random point, X the model identifies the 3 nearest points, based on the euclidean distance. The class label of the new record is the label of the majority of the nearest neighbors, in this case the blue circle, o, would be change for the red star, $*$.

Using GridSearchCV we found that the best parameters from leaf_size = [1,2,...,49]; n_neighbors = [1,2,..,29] and p = [1,2] are [leaf_size=1, n_neighbors=23, p=1] with all the others parameters set to default and being n_neighbors the value of our K.

F. Neural Network (NN)

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and is at the heart of deep learning algorithms however, we have used it as a non-deep classifier. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another [6]. Neural networks are ideally suited to help people solve complex problems in real-life situations. They can learn and model the relationships between inputs and outputs that are nonlinear and complex, make generalizations, reveal hidden relationships, patterns and predictions, and model highly volatile data and variances needed to predict rare events [13]. Neural networks are composed of a certain number of layers, an input and output layer and then, it could have one or more hidden layers. Each layer is composed of set of nodes, the artificial neurons. Each neuron has the ability to let through or not data, depending if his value is above the threshold, letting through the data to the next layer of neurons.
Each node has its own weight assigned and when data meets it, as an input, it gets multiplied by the weight and then summed. If that output exceeds the given threshold the activation function is "fired" and the data is passed to the next layer i.e. the input of a neuron becomes the output of the next node [6].

To play this model the software used was RapidMiner Studio. The first step, shown in the figure 23 , is to apply a numerical to polynomial operator that changes the type of selected numeric attributes to a polynomial type and maps all values of these attributes to corresponding polynomial values. In other words, each numerical value is simply used as nominal value of the new attribute, the condicion. As numerical attributes can have a huge number of different values even in a small range, converting such a numerical attribute to polynomial form will generate a huge number of possible values for the new attribute.

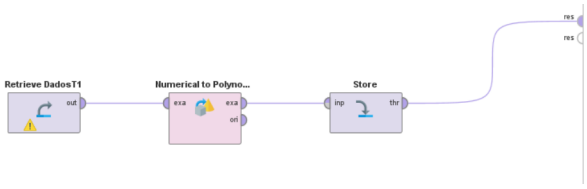

Fig. 23: Neuronal Net.

The next step is represented in figure 24. Firstly is applied a split data operator where we have created a ratio of 80% / 20%, to use it, already mentioned before, 80% of data to train and 20% to test. Neural Net operator is applied and getting the 80% as input. This operator, as well mention above, can have hidden layers yet, since the focus of this course is non-deep strand, we've used only one hidden layer. Each layer has a certain size associated. The number of layers and the number of nodes in each hidden layer are known as hyperparameters and they must be specified. The most reliable way to configure these hyperparameters is via systematic experimentation [4]. Regarding the choice of the number of neurons there are many rule-of-thumb methods for determining the correct number of neurons to use in the hidden layers [8]. In this work, to calculate the number of neurons in the hidden layer, we have used two thirds of the number of neurons in the input layer plus number of neurons in the output layer [8].
Other parameter in the NN operator is the training cycles. This parameter specifies the number of training cycles used for the neural network training. Its default number is two-hundred, but after some testing we figured that four-hundred was the ideal number.

Learning rate, the amount of change to the model during each step [3]. This hyperparameter controls the rate or speed at which the model learns. Typical values for a neural network

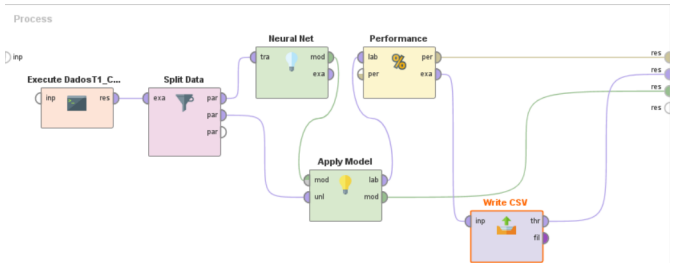

Fig. 24: Neuronal Net.

learning rate are less than 1 and greater than 10^{-6} [3]. On the other hand, a traditional default value for the learning rate is 0.1 or 0.01. After some attempts of different learning rates, the best one was 0.001.

For the momentum, that simply adds a fraction of the previous weight update to the current one leading to an increasing in the size of the steps taken towards the minimum of the gradient, we leave it on zero.

Then, the apply model was applied and ,as the name says, applies the model. It got as input the 20% data and of the splitting data operator and trained model of NN operator. Lastly, we integrate the performance operator. This operator has the function to evaluate the NN model's performance and the performance criteria are automatically determined in order to fit the NN model.

V. Results

A. Confusion Matrix and Performance Metrics

For each model it is possible to evaluate its performance. Every model has a confusion matrix, being the generic form represented in table I.

TABLE I: Confusion Matrix

	True 0	True 1
Pred 0	TP	FN
Pred 1	FP	TN

A confusion matrix is a summary of prediction results on a classification problem [11]. It shows the ways in which our classification model is confused when it makes predictions giving insights, not only into the errors being made, but more importantly the types of errors that are being made. Analysing the matrix we can extract the True Positive (TP), True Negative (TN), False Positive (FP) and the False Negative (FN) values.
For the Logistic Regression model the confusion matrix obtained is represented in table II. For this model we were able to get TP we got 27, for TN we got 26, for FP we got 5 and for FN we got 2.

TABLE II: Confusion Matrix - Logistic Regression

	True 0	True 1
Pred 0	27	2
Pred 1	5	26

For the Support Vector Machine (SVM) model the confusion matrix obtained is represented in table III. For this model we were able to get TP we got 24, for TN we got 24, for FP we got 7 and for FN we got 5.

TABLE III: Confusion Matrix - SVM

	True 0	True 1
Pred 0	24	5
Pred 1	7	24

For the Naive Bayes (Gaussian) model the confusion matrix obtained is represented in table IV. For this model we were able to get TP we got 25, for TN we got 26, for FP we got 5 and for FN we got 4.

TABLE IV: Confusion Matrix - Naive Bayes

	True 0	True 1
Pred 0	25	4
Pred 1	5	26

For the Decision Tree model the confusion matrix obtained is represented in table V. For this model we were able to get TP we got 23, for TN we got 25, for FP we got 6 and for FN we got 6.

TABLE V: Confusion Matrix - Decision Tree

	True 0	True 1
Pred 0	23	6
Pred 1	6	25

For the K-Nearest Neighbour (KNN) model the confusion matrix obtained is represented in table VII. For this model we were able to get TP we got 28, for TN we got 21, for FP we got 4 and for FN we got 6.

TABLE VI: Confusion Matrix - KNN

	True 0	True 1
Pred 0	22	11
Pred 1	8	19

For the Neural Network (NN) model the confusion matrix obtained is represented in table VII. For this model we were able to get TP we got 29, for TN we got 21, for FP we got 3 and for FN we got 6.

TABLE VII: Confusion Matrix - Neural Net

	True 0	True 1
Pred 0	29	6
Pred 1	3	21

With the confusion matrix it is possible to get the accuracy and precision of the model used. For the accuracy, the formula used was:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

To obtain the precision, the formula is:

$$Precision = \frac{TP}{TP + FP}$$

It is also possible to get the F1 Score, a metric that is primarily used to compare the performance of two classifiers, with the confusion matrix. Its formula is:

$$F1 = 2 * \frac{Recall * Precision}{Recall + Precision}$$

where Recall is the TPR.

It is also possible to get the True Positive Rate (TPR), also known as recall or sensitivity, which represents all positive examples the fraction of correctly classified , True Negative Rate (TNR), also known as specificity representing all negative examples the fraction of correctly classified, and False Positive Rate (FPR) that represents how often an actual negative instance will be classified as positive, a fals alarm.

$$TPR = \frac{TP}{TP + FN}$$

$$TNR = \frac{TN}{TN + FP}$$

$$FPR = \frac{FP}{FP + TN}$$

Finally, we can get, by using the confusion matrix, the balanced accuracy, often used when the two classes are imbalanced – that is, one class appears much more than the other. It is represented by the bellow formula:

$$Balanced_{Accuracy} = \frac{Recall + Specificity}{2}$$

Where Specificity is the TNR.

The table VIII show us all of these performance metrics by model.

TABLE VIII: Perfomace Metrics

(%)	Acc	Prec	F1	TPR	TNR	FPR	Bal Acc
LogReg	88.3	84.4	88.5	93.1	83.9	16.1	88.5
SVM	80.0	77.4	79.9	82.8	77.4	22.6	80.0
NB	85.0	83.3	84.7	86.2	83.8	16.1	85.0
DT	80.0	75.9	78.6	81.5	78.8	21.2	80.0
KNN	68.3	73.3	69.8	66.7	70.4	29.6	68.5
NN	84.75	90.63	83.71	77.78	87.50	12.50	82.64

B. Comparing with other articles

We search other articles/works similar to ours to our data set and we notice that when using our data set various authors apply this same models, but using the default parameters. So for that, we applied all this models again but this time using the default parameters, and we obtained the following performance table:

TABLE IX: Perfomace Metrics - Default Parameters

(%)	Acc	Prec	F1	TPR	TNR	FPR	Bal Acc
LogReg	86.7	83.9	86.7	89.7	83.9	16.1	86.8
SVM	61.7	57.1	67.6	82.8	42.0	58.1	62.3
DT	68.3	72.4	68.9	65.6	71.4	28.6	68.5
KNN	66.7	71.0	68.8	66.7	66.7	33.3	66.7
NN	74.6	71.9	75.4	79.3	70.0	30.0	74.7

It's important to notice that all values are lower, as expected, as this aren't the best parameters for our data. Also, as Gaussian Naive Bayes doesn't have any parameters so the results are the same.

C. ROC and AUC curves

The Receiver Operator Characteristic (ROC) curve is an evaluation metric for binary classification problems. It is a probability curve that plots the TPR against FPR at various threshold values and essentially separates the 'signal' from the 'noise'. The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.[3]

When AUC = 1, then the classifier is able to perfectly distinguish between all the Positive and the Negative class points correctly. If, however, the AUC had been 0, then the classifier would be predicting all Negatives as Positives, and all Positives as Negatives. When 0.5<AUC<1, there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values. This is so because the classifier is able to detect more numbers of True positives and True negatives than False negatives and False positives. When AUC=0.5, then the classifier is not able to distinguish between Positive and Negative class points. Meaning either the classifier is predicting random class or constant class for all the data points. [11]

So, the higher the AUC value for a classifier, the better its ability to distinguish between positive and negative classes.

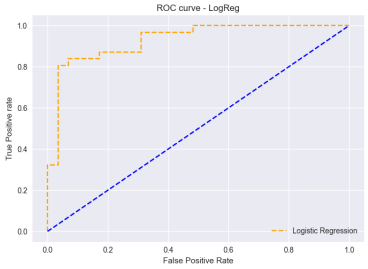

Fig. 25: ROC - Logistic Regression

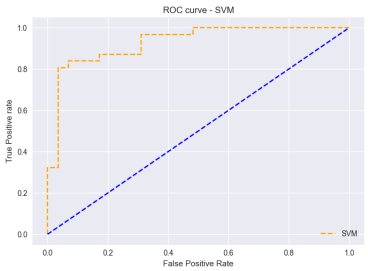

Fig. 26: ROC - SVM

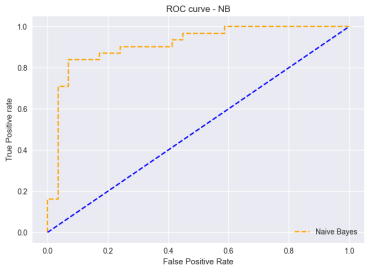

Fig. 27: ROC - Naive Bayes

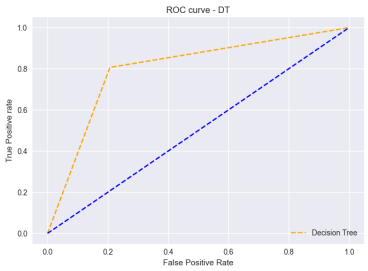

Fig. 28: ROC - Decision Tree

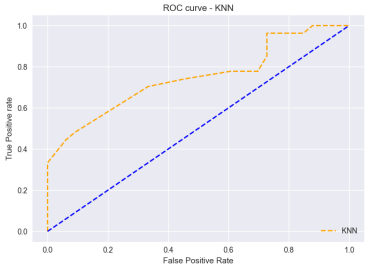

Fig. 29: ROC - K-nearest Neighbor

The models from figures 25, 26, 27, 28 and 29 got an AUC of 0.93,0.93,0.91, 0.80 and 0.75. As the AUC of the first 3 models is very close to 1 we can conclude that the model can perfectly distinguish between all the Positive and the Negative

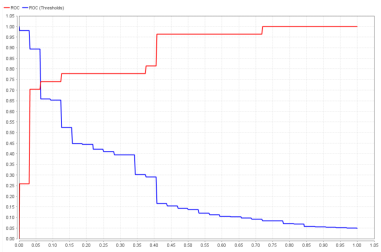

Fig. 30: ROC - Neural Network

class correctly, where as for the Decision Tree and K-nearest Neighbor, although its not perfect, there is a high chance than it can distinguish them correctly.

The ROC graph represented in 30 was made on RapidMiner software and the AUC obteind was 0.89. As it can be seen, it has two lines where the red represents the ROC function and the blue the threshold ROC function. Both line share the same x-axis however their y-axes are different, TPR for ROC and a threshold value for Threshold ROC. The ROC curve is built by decreasing the thresholds, from 1 to 0 and, at each value a confusion matrix is calculated giving a point in the graph (FPR versus TPR) i.e. the ROC graph. Well, the blue curve was built by using the thresholds that were used to build the ROC.

D. Error Functions

To study this functions we just used 2 features, age as X and thalach as Y, this 2 were chosen randomly and we could have selected any of the other features.

1) Loss Function: Loss function is also known as cost function. This function basically means how much far away our predicted line is from the generic data [12]. This function measure the difference between the estimator (the dataset) and the estimated value (the prediction, data after model) [10]. The function used to calculate the cost function was:

$$\mathcal{J}(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2$$

Where m represents the length of the variable in Y axis and h_θ represents the hypothesis. When we applying it, it should be applied for different values of θ so we can see the minimum of the function [10].

2) Gradient Descent: Gradient descent is an optimization algorithm that's used when training a machine learning model. It's based on a convex function and tweaks its parameters iterative to minimize a given function to its local minimum.[7]

Gradient descent algorithm does not work for all functions. There are two specific requirements. A function has to be differentiable and convex. The function used to obtain the gradient for all the models descent is:

$$\frac{\partial \mathcal{J}(\theta)}{\partial \theta} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)x_j^i$$

We also need to see if we are using a good value for the learning rate, in Fig.31 we see the format we want our curve to take and with this we can try different values for the learning rate and see which one applies better.

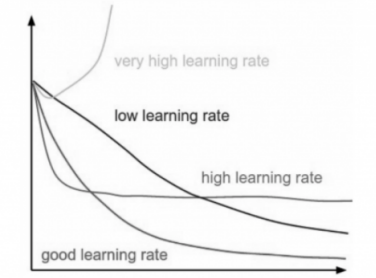

Fig. 31: Good and Bad Learning Rates

After all this requirements we implement the gradient function to our data with an learning rate of 0.01 and we got the following curve.

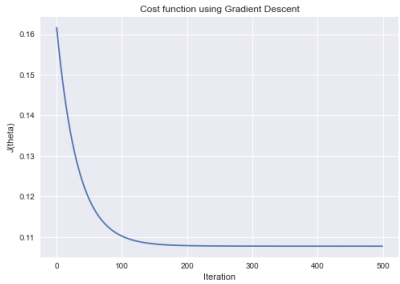

Fig. 32: Gradient Descent, learning rate = 0.01

As we can see just a bit after the 100 iterations our model converges. It is worth noting that to compute the gradient descent it is necessary to choose two attributes. In this case we have 13 of them, so we've chose the attribute age and thalach.

VI. Conclusion

Heart Disease is one of the major concerns for society today. It is difficult to manually determine the odds of getting heart disease based on risk factors. However, machine learning techniques are useful to predict the output from existing data. For that, it's important to get the best prediction, and this only happen when working with the best parameters for the data set in study and with comparing models to see which one gives us a best result.

As seen in chapter V the best model for classification of this data set is the Logistic Regression followed by the Naive Bayes (Gaussian) due to being the models with best accuracy of the test set.

References

[1] Mariette Awad and Rahul Khanna. *Support Vector Machines for Classification*. URL: https : / / www . researchgate . net / publication / 300723807_Support_ Vector_Machines_for_Classification.

[2] Daniel Berrar. *Bayes' Theorem and Naive Bayes Classifier*. URL: https://www.researchgate.net/publication/324933572_Bayes'_Theorem_and_Naive_Bayes_Classifier.

[3] Jason Brownlee. *How to Configure the Learning Rate When Training Deep Learning Neural Networks*. URL: https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/.

[4] Jason Brownlee. *How to Configure the Number of Layers and Nodes in a Neural Network*. URL: https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/.

[5] Padraig Cunningham and Sarah Jane Delany. *k-Nearest neighbour classifiers*. URL: https://www.researchgate.net/publication/228686398_k-Nearest_neighbour_classifiers.

[6] IBM Cloud Education. *Neural Networks*. URL: https://www.ibm.com/cloud/learn/neural-networks.

[7] Sepp Hochreiter and Arthur Steven Younger. *Learning To Learn Using Gradient Descent*. URL: https://www.researchgate.net/publication/225182080_Learning_To_Learn_Using_Gradient_Descent.

[8] Sandhya Krishnan. *How to determine the number of layers and neurons in the hidden layer?* URL: https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3.

[9] Oded Maimon and Lior Rokach. *Decision Trees*. URL: https://www.researchgate.net/publication/222511520_Introduction_to_ROC_analysis.

[10] Lachlan Miller. *Machine Learning week 1: Cost Function, Gradient Descent and Univariate Linear Regression*. URL: https://medium.com/@lachlanmiller_52885/machine-learning-week-1-cost-function-gradient-descent-and-univariate-linear-regression-8f5fe69815fd.

[11] Sarang Narkhede. *Understanding AUC - ROC Curve*. URL: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

[12] Kshitiz Sirohi. *Beginner: Cost Function and Gradient Descent*. URL: https://towardsdatascience.com/machine-leaning-cost-function-and-gradient-descend-75821535b2ef.

[13] SAS - Analytics Software Solutions. *Neural Networks - What they are why they matter*. URL: https://www.sas.com/en_us/insights/analytics/neural-networks.html.

[14] Ravi Verma. *ML Supervised Learning : Logistic Regression Model using Python*. URL: https://www.researchgate.net/publication/343737613_ML_Supervised_Learning_Logistic_Regression_Model_using_Python.