HEART DISEASE PRESENCE PREDICTION

Ilias Chatzistefanidis 2351
ichatsist@uth.gr

Argyris Adam 2309
aradam@uth.gr

Eleni Koutsoni 2371
ekoutsoni@uth.gr

January 22, 2020

ABSTRACT

Our goal in this project is to use different Machine Learning algorithms to decide which are the best in order to predict the presence/absence of a heart disease in a patient.

1 Introduction

Machine Learning (ML) provides methods, techniques, and tools that can help solving diagnostic and prognostic problems in a variety of medical domains. ML is being used for the analysis of the importance of clinical parameters and their combinations for prognosis, e.g. prediction of disease progression, extraction of medical knowledge for outcome research, therapy planning and support, and for the overall patient management.

In this project, we build a heart disease prediction model and make comparison between different methods. The input to our algorithm is Heart Disease UCI data set. We then use traditional ML methods (logistic regression,ridge regression,random forest regression,support vector machine, etc) and two neural networks to predict the presence/absence of heart disease.

2 Relevant Works

Prior works on heart disease presence are deficient in terms of evaluation metrics and performance. Robert Dentaro , Andras Janosi [1989][1] worked on a discriminant function model for estimating probabilities of angiographic coronary disease and tested it for reliability and clinical utility in 3 patient test groups. They compared the results with the results of applying Naive Bayesian algorithm to the same 3 patient test groups. It was concluded that coronary disease probabilities derived from discriminant functions are reliable and clinically useful when applied to patients with chest pain syndromes and intermediate disease prevalence.

Jaymin Patel, Prof. TejalUpadhyay, Dr. Samir Patel [2016][2] compared different algorithms of Decision Tree classification seeking better performance in heart disease diagnosis using WEKA. The algorithms which are tested is J48 algorithm, Logistic model tree algorithm and Random Forest algorithm.The goal of this study is to extract hidden patterns by applying data mining techniques, which are noteworthy to heart diseases and to predict the presence of heart disease in patients where this presence is valued from no presence to likely presence.

3 Dataset

We use the Heart Disease UCI dataset 1988. This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "goal" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4. Experiments with the Cleveland database have concentrated on simply attempting to distinguish presence (values 1,2,3,4) from absence (value 0).

Pre-processing. There was no need for significant pre-processing, because the data set we work on has already been pre-processed and now contains 14 attributes instead of the original 76. In addition these attributes contain only non-null values, so there is no need for cleaning the data. Moreover, since all our attributes have continues/discrete values there is no need to use a transformation function like "LabelEncoder" in order to transform categorical values to numerical.

PCA. We decided to carry out Principal Component Analysis[3] in order to find the best low-dimensional representation of the variation in our dataset. This way, we would like to capture most of the variation between sampled using a smaller number of new variables(principal component), where each of these new variables is a linear combination of all or some of the variables.

Firstly, we standardise the variables using the `scale()` function of *scikit-learn*, which is necessary due to the fact that the variables might have different variances. Then, we carry out a principal component analysis using `PCA` class from *sklearn.decomposition* package and its `fit()` method. Finally, we decide to retain the first eight principal components, because we assumed that it is of great importance at least 75% of the variance to be explained.

4 Methods

We first consider traditional ML methods:

1. Multiple Linear Regression
2. Logistic Regression
3. Ridge Classifier
4. K-Nearest Neighbours Classifier
5. Support Vector Machine (SVM)
6. Kernel SVM
7. Naive Bayes Algorithm
8. Decision Tree Classifier
9. Random Forest Classifier

We then employ deep learning and train two neural networks:

1. MLPClassifier (Multilayer Perceptron Classifier)
2. Keras Neural Network

4.1 Multiple linear regression, Logistic Regression and Ridge Classifier

As a baseline model we run **Multiple Linear Regression** which assumes that the relationship between the dependant variable y an the p-vector of regressors x is linear.

For example, with three predictor variables (x), the prediction of y is expressed by the following equation:

y = b$_0 + b_1 * x_1 + b_2 * x_2 + b_3 * x_3$

The regression beta coefficients measure the association between each predictor variable and the outcome. b_j can be interpreted as the average effect on y of a one unit increase in x_j holding all other predictors fixed.

Since the outcome we're trying to predict only takes values 0 and 1 we'll want to use logistic regression instead of basic linear regression.

Logistic regression measures the relationship between the categorical dependent variable (feature) and one or more independent variables (features) by estimating probabilities using a logistic function, which is the cumulative logistic distribution.

Goal of logistic regression: Predict the "true" proportion of success, , at any value of the predictor.

The logistic function is a sigmoid function, which takes any real input t,(tϵR) , and outputs a value between zero and one; for the logit, this is interpreted as taking input log-odds and having output probability. The standard logistic function $\sigma : \mathbb{R} \to (0, 1)$ is defined as follows:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

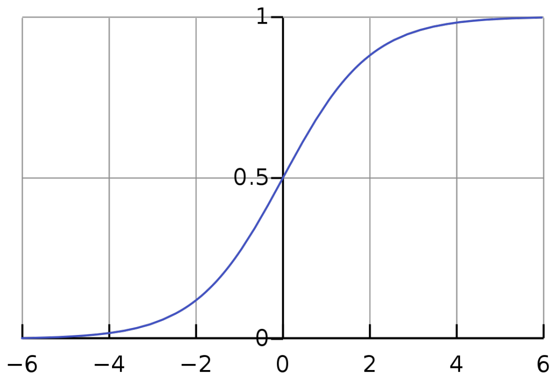


Figure 1: The standard logistic function $\sigma(t)$

Ridge regression is particularly useful to mitigate the problem of multicollinearity in linear regression, which commonly occurs in models with large numbers of parameters. In general, the method provides improved efficiency in parameter estimation problems in exchange for a tolerable amount of bias.

In the simplest case, the problem of a near-singular moment matrix $(\mathbf{X}^\mathsf{T}\mathbf{X})$ is alleviated by adding positive elements to the diagonals. The approach can be conceptualized by posing a constraint $\sum \beta_i^2 = c$ to the least squares problem, such that

$$\min_{\beta} (\mathbf{y} - \mathbf{X}\beta)^\mathsf{T}(\mathbf{y} - \mathbf{X}\beta) + \lambda(\beta^\mathsf{T}\beta - c)$$

where λ is the Lagrange multiplier of the constraint. The minimizer of the problem is the simple ridge estimator

$$\hat{\beta}_R = (\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y}$$

where **I** is the identity matrix and the ridge parameter λ serves as the positive constant shifting the diagonals, thereby decreasing the condition number of the moment matrix.

4.2 K-Nearest Neighbours Classifier

k-Nearest Neighbors (kNN) is a non parametric learning algorithm.

kNN keeps all training examples in memory and thus does not discard the training data after the model is built, like other learning algorithms do. In this way, when a new example x appears, knn finds k training examples nearest to x and returns the majority label in case of a classification, or the average label, in case of regression.

The algorithm calculates the closeness of two examples by a distance function:

1. Euclidean distance: $d(x_i, x_k) = \sqrt{\sum_{j=1}^{D}(x_i^{(j)} - x_k^{(j)})^2}$
2. Negative cosine similarity: $s(x_i, x_k) = \frac{\sum_{j=1}^{D} x_i^{(j)} x_k^{(j)}}{\sqrt{\sum_{j=1}^{D}(x_i^{(j)})^2}\sqrt{\sum_{j=1}^{D}(x_k^{(j)})^2}}$ (cosine similarity) Cosine similarity is a measure of similarity of the directions of two vectors: · If the angle between two vectors is 0 degrees, then two vectors point to the same direction, and cosine similarity is equal to 1. · If the vectors are orthogonal, the cosine similarity is 0. · If vectors are pointing in opposite directions, the cosine similarity is 1. If we want to use cosine similarity as a distance metric,we need to multiply it by 1 (Negative cosine similarity).
3. Chebychev distance
4. Mahalanobis distance
5. Hamming distance
6. learned from data

The choice of the distance metric, as well as the value for k, are the choices the analyst makes before running the algorithm. So these are hyperparameters.

4.3 Support Vector Machine (SVM) Algorithm, Kernel SVM

Support Vector Machines[4] are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training samples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new test samples to one category or the other, making it a non-probabilistic binary linear classifier.

The basic mathematics behind the SVM is however less familiar to most of us. It relies on the definition of hyperplanes and the definition of a margin which separates classes (in case of classification problems) of variables. It is also used for regression problems.

With SVMs we distinguish between hard margin and soft margins. The latter introduces a so-called softening parameter. We distinguish also between linear and non-linear approaches. The latter are the most frequent ones since it is rather unlikely that we can separate classes easily by say straight lines.

SVM algorithms use a set of mathematical functions that are defined as the **kernel**. The function of kernel is to take data as input and transform it into the required form. Different SVM algorithms use different types of kernel functions. These functions can be different types.

For example linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid.

Introduce Kernel functions for sequence data, graphs, text, images, as well as vectors. The most used type of kernel function is RBF. Because it has localized and finite response along the entire x-axis. The kernel functions return the inner product between two points in a suitable feature space. Thus by defining a notion of similarity, with little computational cost even in very high-dimensional spaces.

$$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$$

Polynomial kernel equation

Figure 2: Polynomial kernel equation

$$k(x, y) = \exp\left(-\frac{||x-y||^2}{2\sigma^2}\right)$$

Gaussian kernel equation

Figure 3: Gaussian kernel equation

$$k(x, y) = \tanh(\alpha x^T y + c)$$

Sigmoid kernel equation

Figure 4: Sigmoid kernel equation

4.4 Naive Bayes Algorithm

Naïve Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features.

Bayes' Rule (Bayes' Theorem):

The conditional probability Pr(X = x|Y = y) [5] is the probability of the random variable X to have a specific value x given that another random variable Y has a specific value of y. The Bayes' Rule stipulates that:

$$\Pr(X = x|Y = y) = \frac{Pr(Y=y|X=x)Pr(X=x)}{Pr(Y=y)}$$

4.5 Decision Tree Classifier, Random Forest Classifier

A **decision tree** is an acyclic graph [5] that can be used to make decisions.

In each branching node of the graph, a specific feature j of the feature vector is examined: · If the value of the feature is below a specific threshold, then the left branch is followed. · If the value of the feature is equal or more than a specific threshold, then the right branch is followed. As the leaf node is reached, the decision is made about the class to which the example belongs. a decision tree can be learned from data. A formulation of the decision tree learning algorithm id ID3. The optimization criterion, in this case, is the average log-likelihood:

$$\frac{1}{N}\sum_{i=1}^{N} y_i ln f_{ID3}(x_i) + (1-y_i) ln(1-f_{ID3}(x_i)) \,, \qquad f_{ID3} \text{ is a decision tree}$$

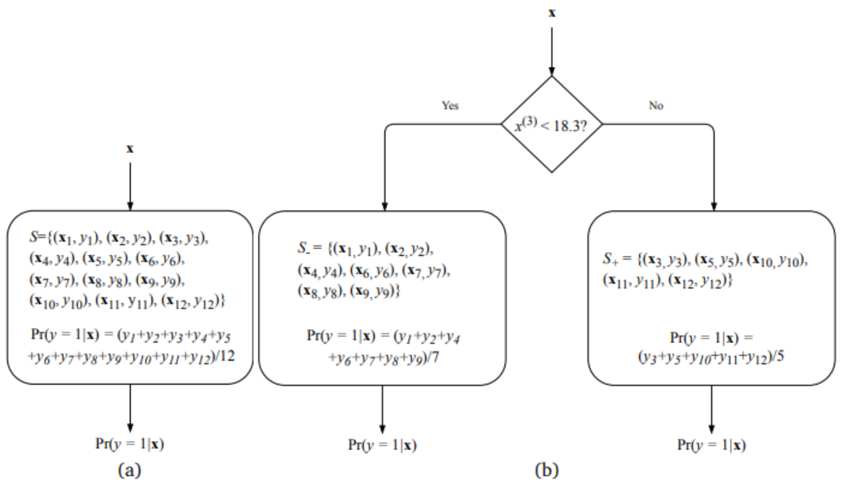

x
Yes
No
$x^{(3)} < 18.3?$
x
$S=\{(\mathbf{x}_1,y_1),(\mathbf{x}_2,y_2),(\mathbf{x}_3,y_3),(\mathbf{x}_4,y_4),(\mathbf{x}_5,y_5),(\mathbf{x}_6,y_6),(\mathbf{x}_7,y_7),(\mathbf{x}_8,y_8),(\mathbf{x}_9,y_9),(\mathbf{x}_{10},y_{10}),(\mathbf{x}_{11},y_{11}),(\mathbf{x}_{12},y_{12})\}$
$\Pr(y=1|\mathbf{x}) = (y_1+y_2+y_3+y_4+y_5+y_6+y_7+y_8+y_9+y_{10}+y_{11}+y_{12})/12$
$S_- = \{(\mathbf{x}_1,y_1),(\mathbf{x}_2,y_2),(\mathbf{x}_4,y_4),(\mathbf{x}_6,y_6),(\mathbf{x}_7,y_7),(\mathbf{x}_8,y_8),(\mathbf{x}_9,y_9)\}$
$\Pr(y=1|\mathbf{x}) = (y_1+y_2+y_4+y_6+y_7+y_8+y_9)/7$
$S_+ = \{(\mathbf{x}_3,y_3),(\mathbf{x}_5,y_5),(\mathbf{x}_{10},y_{10}),(\mathbf{x}_{11},y_{11}),(\mathbf{x}_{12},y_{12})\}$
$\Pr(y=1|\mathbf{x}) = (y_3+y_5+y_{10}+y_{11}+y_{12})/5$
$\Pr(y=1|\mathbf{x})$
$\Pr(y=1|\mathbf{x})$
$\Pr(y=1|\mathbf{x})$
(a)
(b)

Figure 3.4: An illustration of a decision tree building algorithm. The set $\mathcal{S}$ contains 12 labeled examples. (a) In the beginning, the decision tree only contains the start node; it makes the same prediction for any input. (b) The decision tree after the first split; it tests whether feature 3 is less than 18.3 and, depending on the result, the prediction is made in one of the two leaf nodes.

The "vanilla" bagging algorithm works like follows.

Given a training set, we create B random samples Sb (for each b = 1,...,B) of the training set and build a decision tree model f_b using each sample S_b as the training set. To sample S_b for some b, we do the sampling with replacement. This means that we start with an empty set, and then pick at random an example from the training set and put its exact copy to S_b by keeping the original example in the original training set. We keep picking examples at random until the $|S_b|$ = N.

After training, we have B decision trees. The prediction for a new example x is obtained as the average of B predictions:

$$\text{y} \leftarrow \hat{f}(\text{x}) = \frac{1}{B}\sum_{b=1}^{B} f_b(x),$$

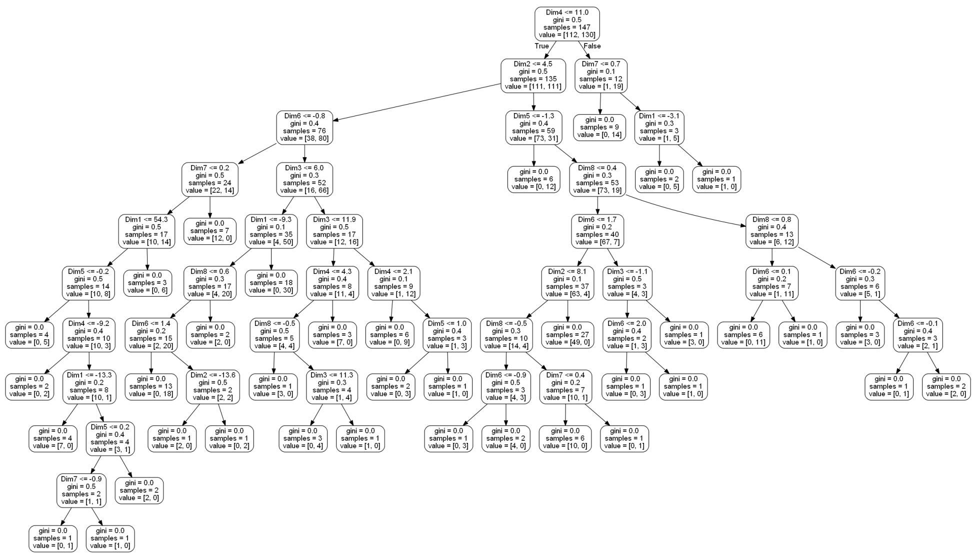
in the case of regression, or by taking the majority vote in the case of classification.

Random forest[6] is different from the vanilla bagging in just one way. It uses a modified tree learning algorithm that inspects, at each split in the learning process, a random subset of the features.

The reason for doing this is to avoid the correlation of the trees: if one or a few features are very strong predictors for the target, these features will be selected to split examples in many trees. This would result in many correlated trees in our "forest." Correlated predictors cannot help in improving the accuracy of prediction. The main reason behind a better performance of model ensembling is that models that are good will likely agree on the same prediction, while bad models will likely disagree on different ones. Correlation will make bad models more likely to agree, which will hamper the majority vote or the average.

The most important hyperparameters to tune are the number of trees, B, and the size of the random subset of the features to consider at each split.

Random forest is one of the most widely used ensemble learning algorithms[5]. Why is it so effective? The reason is that by using multiple samples of the original dataset, we reduce the variance of the final model. Remember that the low variance means low overfitting.


Dim4 <= 11.0 gini = 0.5 samples = 147 value = [112, 130]
True
False
Dim2 <= 4.5 gini = 0.5 samples = 135 value = [111, 111]
Dim7 <= 0.7 gini = 0.1 samples = 12 value = [1, 19]
Dim6 <= -0.8 gini = 0.4 samples = 76 value = [38, 80]
Dim5 <= -1.3 gini = 0.4 samples = 59 value = [73, 31]
gini = 0.0 samples = 9 value = [0, 14]
Dim1 <= -3.1 gini = 0.3 samples = 3 value = [1, 5]
Dim7 <= 0.2 gini = 0.5 samples = 24 value = [22, 14]
Dim3 <= 6.0 gini = 0.3 samples = 52 value = [16, 66]
gini = 0.0 samples = 6 value = [0, 12]
Dim8 <= 0.4 gini = 0.3 samples = 53 value = [73, 19]
gini = 0.0 samples = 2 value = [0, 5]
gini = 0.0 samples = 1 value = [1, 0]
Dim1 <= 54.3 gini = 0.5 samples = 17 value = [10, 14]
gini = 0.0 samples = 7 value = [12, 0]
Dim1 <= -9.3 gini = 0.1 samples = 35 value = [4, 50]
Dim3 <= 11.9 gini = 0.5 samples = 17 value = [12, 16]
Dim6 <= 1.7 gini = 0.2 samples = 40 value = [67, 7]
Dim8 <= 0.8 gini = 0.4 samples = 13 value = [6, 12]
Dim5 <= -0.2 gini = 0.5 samples = 14 value = [10, 8]
gini = 0.0 samples = 3 value = [0, 6]
Dim8 <= 0.6 gini = 0.3 samples = 17 value = [4, 20]
gini = 0.0 samples = 18 value = [0, 30]
Dim4 <= 4.3 gini = 0.4 samples = 8 value = [11, 4]
Dim4 <= 2.1 gini = 0.1 samples = 9 value = [1, 12]
Dim2 <= 8.1 gini = 0.1 samples = 37 value = [63, 4]
Dim3 <= -1.1 gini = 0.5 samples = 3 value = [4, 3]
Dim6 <= 0.1 gini = 0.2 samples = 7 value = [1, 11]
Dim6 <= -0.2 gini = 0.3 samples = 6 value = [5, 1]
gini = 0.0 samples = 4 value = [0, 5]
Dim4 <= -9.2 gini = 0.4 samples = 10 value = [10, 3]
Dim6 <= 1.4 gini = 0.2 samples = 15 value = [2, 20]
gini = 0.0 samples = 2 value = [2, 0]
Dim8 <= -0.5 gini = 0.5 samples = 5 value = [4, 4]
gini = 0.0 samples = 3 value = [7, 0]
gini = 0.0 samples = 6 value = [0, 9]
Dim5 <= 1.0 gini = 0.4 samples = 3 value = [1, 3]
Dim8 <= -0.5 gini = 0.3 samples = 10 value = [14, 4]
gini = 0.0 samples = 27 value = [49, 0]
Dim6 <= 2.0 gini = 0.4 samples = 2 value = [1, 3]
gini = 0.0 samples = 1 value = [3, 0]
gini = 0.0 samples = 6 value = [0, 11]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 3 value = [3, 0]
Dim6 <= -0.1 gini = 0.4 samples = 3 value = [2, 1]
gini = 0.0 samples = 2 value = [0, 2]
Dim1 <= -13.3 gini = 0.2 samples = 8 value = [10, 1]
gini = 0.0 samples = 13 value = [0, 18]
Dim2 <= -13.6 gini = 0.5 samples = 2 value = [2, 2]
gini = 0.0 samples = 1 value = [3, 0]
Dim3 <= 11.3 gini = 0.3 samples = 4 value = [1, 4]
gini = 0.0 samples = 2 value = [0, 3]
gini = 0.0 samples = 1 value = [1, 0]
Dim6 <= -0.9 gini = 0.5 samples = 3 value = [4, 3]
Dim7 <= 0.4 gini = 0.2 samples = 7 value = [10, 1]
gini = 0.0 samples = 1 value = [0, 3]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 2 value = [2, 0]
gini = 0.0 samples = 4 value = [7, 0]
Dim5 <= 0.2 gini = 0.4 samples = 4 value = [3, 1]
gini = 0.0 samples = 1 value = [2, 0]
gini = 0.0 samples = 1 value = [0, 2]
gini = 0.0 samples = 3 value = [0, 4]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 1 value = [0, 3]
gini = 0.0 samples = 2 value = [4, 0]
gini = 0.0 samples = 6 value = [10, 0]
gini = 0.0 samples = 1 value = [0, 1]
Dim7 <= -0.9 gini = 0.5 samples = 2 value = [1, 1]
gini = 0.0 samples = 2 value = [2, 0]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 1 value = [1, 0]

4.6 Multilayer Perceptron Classifier, Keras Neural Network

A single layer perceptron can solve simple problems where data is linearly separable in to 'n' dimensions, where 'n' is the number of features in the dataset. However, in case of non-linearly separable data, the accuracy of single layer perceptron decreases significantly. **Multilayer perceptrons**[7], on the other hand, can work efficiently with non-linearly separable data.

Multilayer perceptrons, or more commonly referred to as artificial neural networks, are a combination of multiple neurons connected in the form a network. An artificial neural network has an input layer, one or more hidden layers, and an output layer. This is shown in the image below:

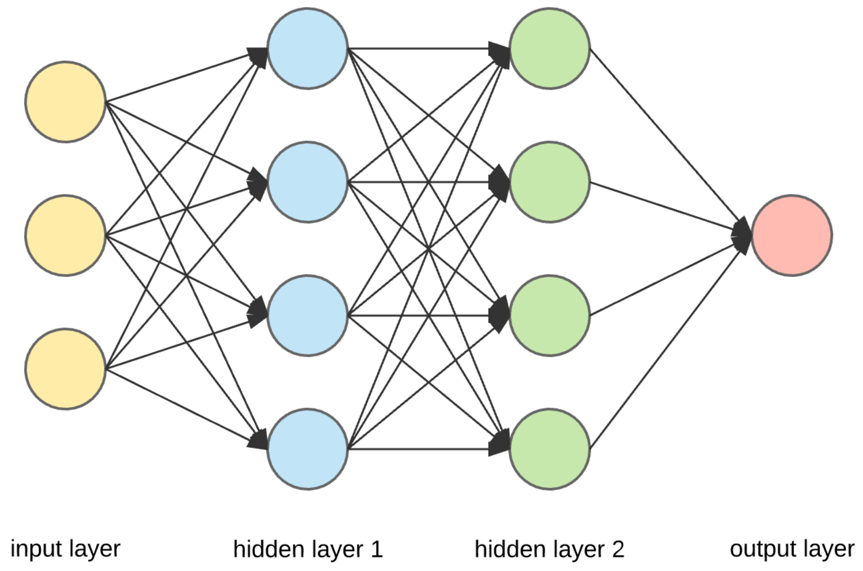


Figure 5: Multilayer Perceptron

Keras is a powerful easy-to-use Python library for developing and evaluating deep learning models[7]. It wraps the efficient numerical computation libraries Theano and TensorFlow and allows you to define and train neural network models in a few short lines of code.

The hidden layer will use the sigmoid function for activations.

The output layer has only one node and is used for the regression, the output of the node is the same as the input of the node. That is, the activation function is $f(x) = x$. A function that takes the input signal and generates an output signal, but takes into account the threshold, is called an activation function.

We work through each layer of our network calculating the outputs for each neuron. All of the outputs from one layer become inputs to the neurons on the next layer. This process is called **forward propagation**.

We use the weights to propagate signals forward from the input to the output layers in a neural network. We use the weights to also propagate error backwards from the output back into the network to update our weights. This is called **backpropagation**.

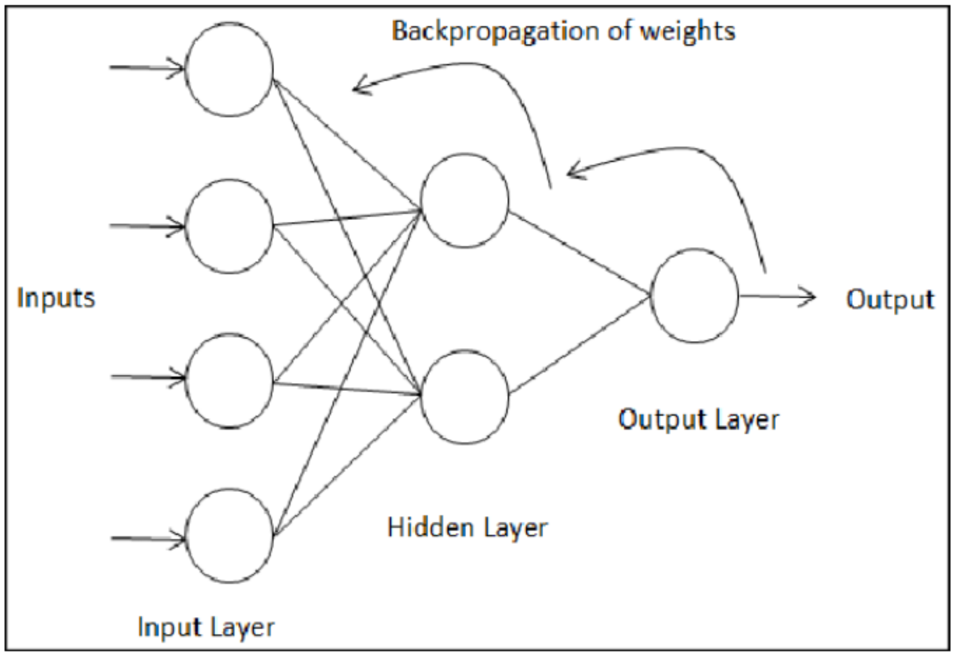


Figure 6: Backpropagation

5 Experiments and Results

Since we have split our dataset into training and testing parts, we use the training part to fit and train our model and then we use the test part to make predictions and check the accuracy of each of the methods we apply. We also store the accuracy of the training part in order to make conclusions on which methods are more stable.

5.1 Results

5.1.1 Based on training accuracy

```
                            0           1
0   Multiple Linear Regression   46.653726
1          Logistic Regression   84.297521
2             Ridge Classifier   83.880000
3                         k-NN   66.528926
4                          SVM  100.000000
5                     SVM Poly   99.586777
6                    SVM Gauss  100.000000
7                     SVM Sigm   51.652893
8                  Naive Bayes   79.752066
9                Decision Tree  100.000000
10               Random Forest  100.000000
11                      NN MLP   93.801653
12                    NN Keras   85.537190
```

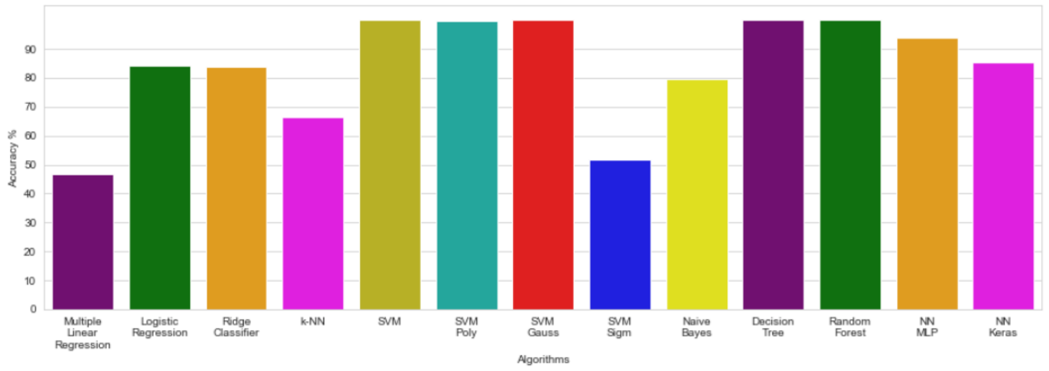


Figure 7: Based on training accuracy

5.1.2 Based on testing accuracy

```
                            0          1
0   Multiple Linear Regression  48.650586
1          Logistic Regression  88.524590
2             Ridge Classifier  88.520000
3                         k-NN  75.409836
4                          SVM  52.459016
5                     SVM Poly  72.131148
6                    SVM Gauss  52.459016
7                     SVM Sigm  55.737705
8                  Naive Bayes  88.524590
9                Decision Tree  73.770492
10               Random Forest  80.327869
11                      NN MLP  80.327869
12                    NN Keras  90.163934
```

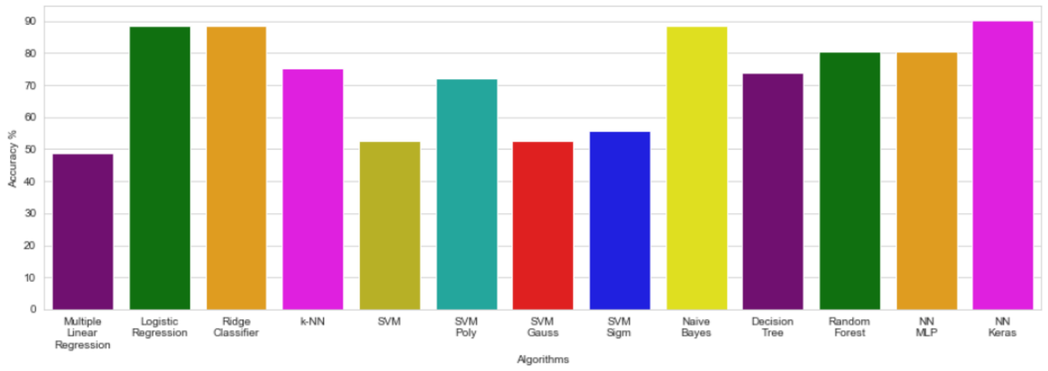
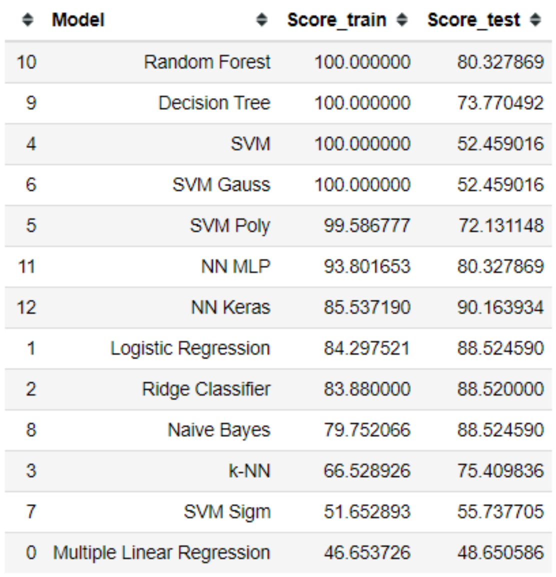


Figure 8: Based on testing accuracy

5.1.3 Comparing based on metrics: score_train, score_test, score_diff

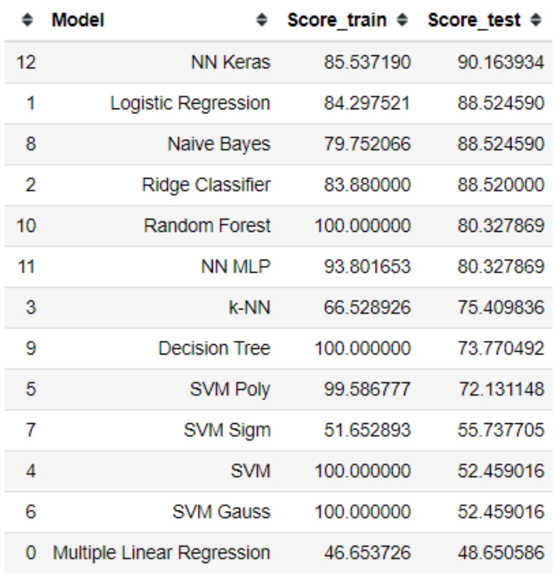
We print the score_train and score_test lists of all methods in descending order of score_train. In this way we can easily observe which models are best suited to the training data.



	Model	Score_train	Score_test
10	Random Forest	100.000000	80.327869
9	Decision Tree	100.000000	73.770492
4	SVM	100.000000	52.459016
6	SVM Gauss	100.000000	52.459016
5	SVM Poly	99.586777	72.131148
11	NN MLP	93.801653	80.327869
12	NN Keras	85.537190	90.163934
1	Logistic Regression	84.297521	88.524590
2	Ridge Classifier	83.880000	88.520000
8	Naive Bayes	79.752066	88.524590
3	k-NN	66.528926	75.409836
7	SVM Sigm	51.652893	55.737705
0	Multiple Linear Regression	46.653726	48.650586

Figure 9: Score train comparison

We print the score_train and score_test lists of all methods in descending order of score_test. In this way we can easily observe which models have best learned how to predict test or other data

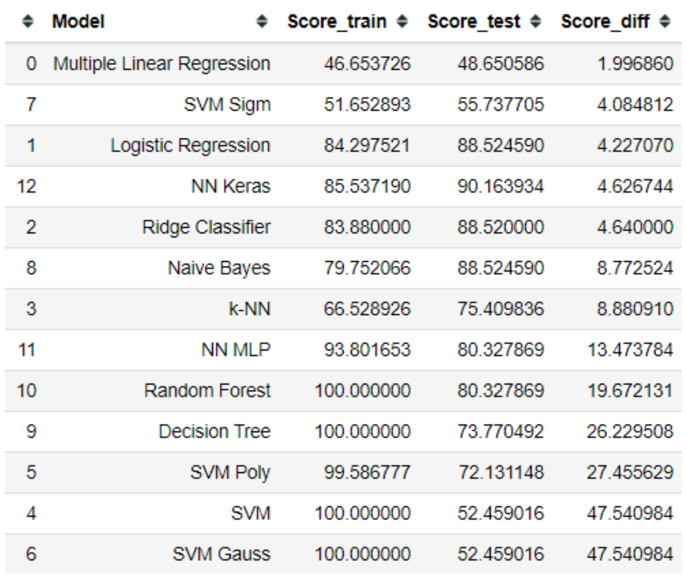


	Model	Score_train	Score_test
12	NN Keras	85.537190	90.163934
1	Logistic Regression	84.297521	88.524590
8	Naive Bayes	79.752066	88.524590
2	Ridge Classifier	83.880000	88.520000
10	Random Forest	100.000000	80.327869
11	NN MLP	93.801653	80.327869
3	k-NN	66.528926	75.409836
9	Decision Tree	100.000000	73.770492
5	SVM Poly	99.586777	72.131148
7	SVM Sigm	51.652893	55.737705
4	SVM	100.000000	52.459016
6	SVM Gauss	100.000000	52.459016
0	Multiple Linear Regression	46.653726	48.650586

Figure 10: Score test comparison

We can check which models are more stable by finding the minimum remainder of the following equation :

$$||(model[Score\_train] - model[Score\_test])||$$



	Model	Score_train	Score_test	Score_diff
0	Multiple Linear Regression	46.653726	48.650586	1.996860
7	SVM Sigm	51.652893	55.737705	4.084812
1	Logistic Regression	84.297521	88.524590	4.227070
12	NN Keras	85.537190	90.163934	4.626744
2	Ridge Classifier	83.880000	88.520000	4.640000
8	Naive Bayes	79.752066	88.524590	8.772524
3	k-NN	66.528926	75.409836	8.880910
11	NN MLP	93.801653	80.327869	13.473784
10	Random Forest	100.000000	80.327869	19.672131
9	Decision Tree	100.000000	73.770492	26.229508
5	SVM Poly	99.586777	72.131148	27.455629
4	SVM	100.000000	52.459016	47.540984
6	SVM Gauss	100.000000	52.459016	47.540984

Figure 11: Stability of algorithms comparison

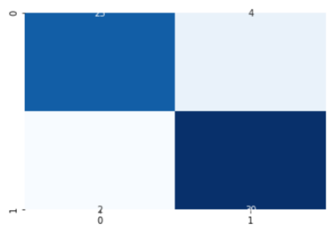
5.2 Conclusions

From the above results we observe that in some models the accuracy of the training is more than the accuracy of testing, this is called overfitting. In this case, we can not trust this model in the classification. Thus, taking into consideration the testing accuracy and minimum reminder of our stability equation(|score_train - score_test|) we are able to decide which models are the best for our data.

We conclude that the best three models are :

Neural Network Keras

```
[[25  4]
 [ 2 30]]
```



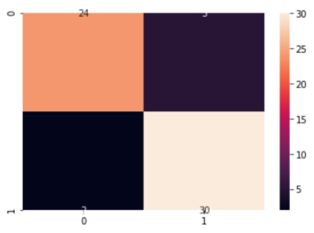
```
accuracy of the model:  90.1639344262295
              precision    recall  f1-score   support

           0       0.93      0.86      0.89        29
           1       0.88      0.94      0.91        32

    accuracy                           0.90        61
   macro avg       0.90      0.90      0.90        61
weighted avg       0.90      0.90      0.90        61
```

Logistic Regression

```
[[24  5]
 [ 2 30]]
```



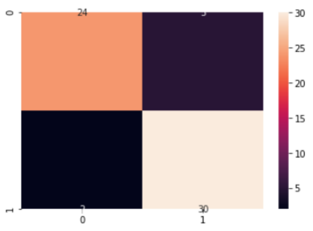
```
              precision    recall  f1-score   support

           0       0.92      0.83      0.87        29
           1       0.86      0.94      0.90        32

    accuracy                           0.89        61
   macro avg       0.89      0.88      0.88        61
weighted avg       0.89      0.89      0.88        61
```

Ridge Classifier

```
[[24  5]
 [ 2 30]]
```



```
              precision    recall  f1-score   support

           0       0.92      0.83      0.87        29
           1       0.86      0.94      0.90        32

    accuracy                           0.89        61
   macro avg       0.89      0.88      0.88        61
weighted avg       0.89      0.89      0.88        61
```

References

[1] Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., Schmid, J., Sandhu, S., Guppy, K., Lee, S., Froelicher, V. (1989). International application of a new probability algorithm for the diagnosis of coronary artery disease. American Journal of Cardiology, 64,304–310.

[2] Jaymin Patel, Prof.TejalUpadhyay, Dr. Samir Patel(2016). Heart Disease Prediction Using Machine learning and Data Mining Technique.

[3] S. Valle, W. Li, S.J. Qin Selection of the number of principal components: the variance of the reconstruction error criterion with a comparison to other methods Industrial and Engineering Chemistry Research, 38 (1999), pp. 4389-4401

[4] N. Christianini, J. Shawe-Taylor An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods, Cambridge university press, UK (2000)

[5] Burkov, A.: The Hundred-Page Machine Learning Book, Quebec (2019)

[6] L. Breimann: Random forests Mach. Learn., 45 (2001), pp. 5-32

[7] Scott Robinson: Introduction to Neural Networks with Scikit-Learn, January 29, 2018