UNIVERSITY OF OSLO

Project 3

Data Analysis and Machine Learning

FYS-STK3155/4155

Isabella Rositi

December 17, 2021

Abstract

An essential part of analyzing data is the ability of making accurate predictions, which often requires specific models that fit best this purpose. In this project, as in the previous Project 2, the aim is to find out which model can best predict the data, applying a Regression Model, Neural Networks, Decision Trees and Ensemble Methods. I will investigate a classification problem about whether the passengers on the Titanic have survived or not, using as features their age, their passenger-class, their sex and the fare they paid. I will split the dataset into training and test set in order to make predictions, so that the model can be trained on specific observations and then tested on new instances. By doing so the results are going to be more significant. I will take into consideration different methods, namely Logistic Regression, Neural Networks, Decision Trees and Ensemble Methods, such as Bagging, Random Forests and Boosting. Among these models the Neural Network with 2 hidden layers and ReLU as activation function turned out to be the one showing the higher *Accuracy Score,* suggesting it works better than other methods when making predictions.

1. Introduction

Data is an extremely important tool and it can offer precious insights on everything nowadays, but using it correctly to gain useful information is not as intuitive as it might seem. Researchers in the Statistical field have studied and developed numerous methods that can be implemented when performing an analysis, but how to choose which one? This is one of the main problems everybody has to face when dealing with data: there is not a ready-made answer.

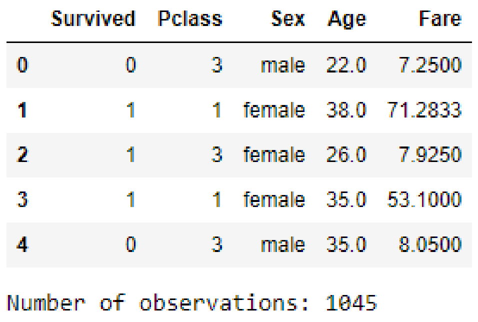
In this project, as in Project 2, the aim is to define the best model for this specific study, namely for making predictions. In fact, models perform differently based on the purpose of the analysis, and the results shown here have to be interpreted keeping this important aspect in mind. I am going to work with a dataset containing information about the Titanic, where the response variable "Survived" is dichotomous and represents whether the passengers have survived or not to the sinking. Each row represents one passenger, and the columns describe different attributes about the person.
The analysis starts with two methods that I have also implemented in Project 2: Logistic Regression with and without regularization and Neural Networks with different numbers of hidden layers, hidden nodes and different activation functions. Moreover, in this project I will go over Decision Trees and Ensemble Methods, specifically Bagging, Random Forests and Boosting (Gradient Boosting and AdaBoost). The question this project aims to answer is: Which model is the best?
Since I am interested in prediction making, I use the *Accuracy Score* as cost function and the goal is to maximize it, hence maximize the percentage of accurate predictions.

Now I will describe all the methods I use in this project, then I will present the salient outputs and respective analysis followed by the conclusion.

2. Methods

2.1 Data: Titanic

The dataset I will be using to carry out the analysis contains information about the Titanic, and the focus is on the response variable *Survived,* which is dichotomous and it represents whether a passenger has survived the sinking or not and it is explained by four features: *Age, Sex, Pclass* and *Fare*. *Sex* and *Pclass* are categorical features, respectively with two and three categories, so I encoded them using the one-hot method. The final design matrix has dimensions (1045x7).



	Survived	Pclass	Sex	Age	Fare
0	0	3	male	22.0	7.2500
1	1	1	female	38.0	71.2833
2	1	3	female	26.0	7.9250
3	1	1	female	35.0	53.1000
4	0	3	male	35.0	8.0500

```
Number of observations: 1045
```

Table 1: Titanic dataset, original format

For the purpose of the analysis it is best to scale the data by standardizing the variables. By doing so, the range of the variance decreases immensely and the predictions result more accurate. For instance, the variance of the variable *Age* goes from 207.84 when data is not scaled, to 0.70.

2.2 Logistic Regression

Logistic regression deals with classification problems, where the response variable is categorical and not continuous. The most common situation is encountered when the response variable presents only two possible outcomes, normally denoted as a binary or dichotomous outcome. This is the case of the problem presented in this project.

In logistic regression the probability that a data point x_i belongs to a category $y_i = \{0,1\}$ is given by the so-called logit function, which is meant to represent the likelihood for a given event.

Let's define some quantities.

$$odds = \frac{p}{1-p} \qquad [1]$$

represents the probability of success over the probability of failure,

$$logit = log(odds) = log\left(\frac{p}{1-p}\right) \qquad [2]$$

which is the logarithm of the odds,

$$expit = \frac{e^{z_i}}{1+e^{z_i}} \qquad [3]$$

which is the inverse of the logit and represents the probability of success.

From now on let's assume that the classification problem has two classes with y_i either 0 or 1, so the probabilities are:

$$p(y_i = 1|\beta x) = \frac{\exp(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p)}{1 + \exp(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p)} \qquad [4]$$

$$p(y_i = 0|\beta x) = 1 - p(y_i = 1|\beta x) \qquad [5]$$

I will use the *Maximum Likelihood Estimation (MLE) principle* in order to define the total likelihood for all possible outcomes from a dataset $D = \{x_i, y_i\}$ with the binary outcome $y_i \in \{0,1\}$ and independently drawn data points. The aim is to maximize the probability of sampling the observed data.

Then let's approximate the likelihood in terms of the product of the individual probabilities of a specific outcome y_i, that is:

$$P(D|\beta) = \prod_{i=1}^{n} [p(y_i = 1|\beta x)]^{y_i} \left[1 - p(y_i = 1|\beta x)\right)]^{1-y_i} \qquad [6]$$

from which it is obtainable the log-likelihood and the cost function.

The cost function being:

$$C(\beta) = \sum_{i=1}^{n}(y_i \ log[p(y_i = 1|\beta x)] + (1 - y_i) \ log[1 - p(y_i = 1|\beta x))]) =$$

$$= \sum_{i=1}^{n}(y_i(\beta x) - \log (1 + \exp (\beta x))) \qquad [7]$$

The maximum likelihood estimator is defined as the set of parameters that maximize the log-likelihood with respect to β. Since the cost function is just the negative log-likelihood, for logistic regression let's define the cross entropy:

$$C(\beta) = -\sum_{i=1}^{n} \left(y_i(\beta x) - log\left(1 + exp(\beta x)\right)\right) \qquad [8]$$

The cross entropy is a convex function of the weights β and, therefore, any local minimizer is a global minimizer.

Minimizing this cost function with respect to β_j gives:

$$\frac{\partial C(\beta)}{\partial \beta_j} = -\sum_{i=1}^{n}(y_j x_j - x_j \frac{\exp(\beta x)}{1 + \exp(\beta x)}) \qquad [9]$$

which can be rewritten using a more compact form as:

$$\frac{\partial C(\beta)}{\partial \beta} = -X^T(y - p) \qquad [10]$$

In addition, let's define a diagonal matrix W with elements $p(y_i|\beta x)(1 - p(y_i|\beta x)$, so that a new compact expression of the second derivative can be:

$$\frac{\delta^2 C(\beta)}{\delta \beta \delta \beta^T} = X^T W X \qquad [11]$$

This defines the Hessian matrix.

To solve these equations, the Newton-Raphson's iterative method is normally the method of choice. It requires however that the matrices that define the first and second derivatives can be computed in an efficient way, so there should not be a case of high-dimensionality. The iterative scheme is then given by:

$$\beta_{new} = \beta_{old} - (X^T W X)^{-1} \left(-X^T (y - p)\right)_{\beta_{old}} \qquad [12]$$

The right-hand side is computed with the old values of β.

Using logistic regression without any additional regularization does not always guarantee a good result because of over-fitting. This happens especially when the number of observations of the training set is not large enough, compared to the number of features. In order to get a better classifier, a regularization parameter needs to be added. In this project the parameters of logistic regression have been penalized through λ in such way:

$$\beta \leftarrow \beta - \eta \left(g + \frac{\lambda\beta}{n}\right) \qquad [13]$$

2.3 Neural Networks

An Artificial Neural Network (ANN) is a computational non-linear model for supervised learning that consists of layers of connected neurons (or nodes). It is supposed to mimic a biological nervous system by letting each neuron interact with other neurons by sending signals in the form of mathematical functions between different layers. A wide variety of ANNs have been developed, but most of them consist of an input layer, eventual in-between layers (hidden layers) and an output layer. All layers can contain an arbitrary number of nodes, each connection between two nodes is associated with a weight variable and each time the information goes from a layer to another there is a threshold (bias) that needs to be reached in order for the signal to move forward. There are several types of Neural Networks, but the one implemented in this project is a feed-forward neural network. In this network the information moves in only one direction: forward through the layers, from the input to the output.

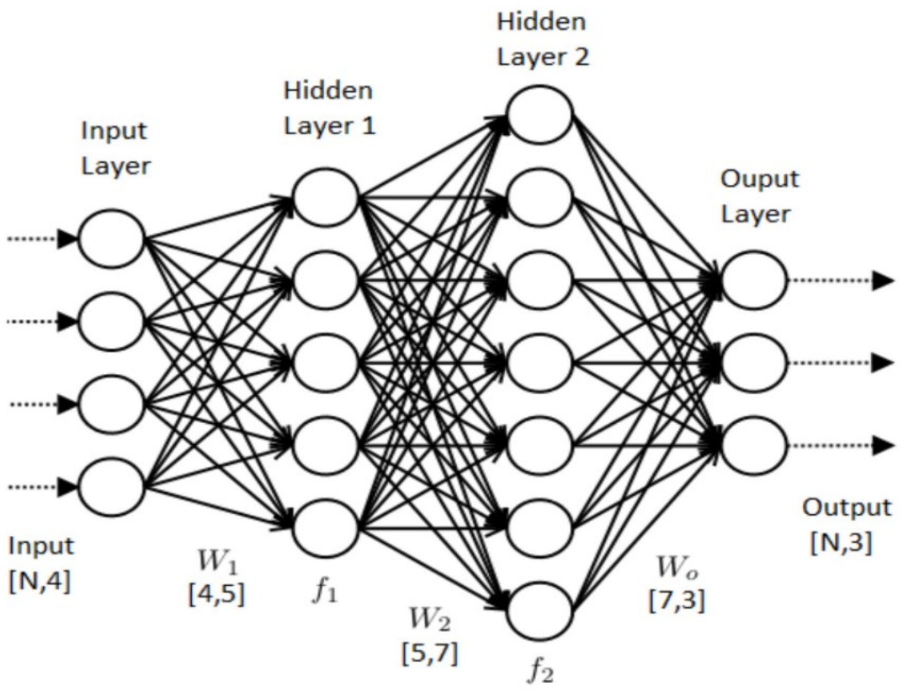


Figure 1: Neural Network with two hidden layers

Nodes are represented by circles, while the arrows display the connections between the nodes, including the direction of the information flow. Additionally, each arrow corresponds to a weight variable. It is observable that each node in a layer is connected to all nodes in the subsequent layer, making this a so-called fully-connected FFNN. One uses often the fully-connected feed-forward neural networks with three or more layers (an input layer, one or more hidden layers and an output layer) consisting of neurons that have non-linear activation functions. Such networks are often called multilayer perceptrons (MLPs).

2.4.1 Mathematical Model

The output y is produced via the activation function f:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b_i\right) = f(z) \qquad [14]$$

Where w_i is the weight for every neuron and b_i is the bias. Usually the bias is kept constant throughout the whole layer. This function receives x_i as inputs. In a FFNN the inputs x_i are the outputs from the previous layer.

Let's start considering the first hidden layer: for each node i is calculated a weighted sum z_i^1 of the input coordinates x_j:

$$z_i^1 = \sum_{j=1}^{M} w_{ij}^1 x_j + b_i^1 \qquad [15]$$

Where M is the total number of possible inputs to the i-th node and the superscript 1 refers to the first hidden layer. The value of z_i^1 is the argument of the activation function f_i of each node i and the output is a_i^1:

$$a_i^1 = f(z_i^1) = f\left(\sum_{j=1}^{M} w_{ij}^1 x_j + b_i^1\right) \qquad [16]$$

where it is assumed that all nodes in the same layer have identical activation functions.

The output of neuron i in layer 2 is then:

$$a_i^2 = f^2(z_i^2) = f^2\left(\sum_{j=1}^{N} w_{ij}^2 y_j^1 + b_i^2\right) = f^2\left[\sum_{j=1}^{N} w_{ij}^2 f^1\left(\sum_{k=1}^{M} w_{jk} x_k + b_j^1\right) + b_i^2\right] \qquad [17]$$

And so on for every layer.

The generalized expression for an MLP with l hidden layers is:

$$a_i^{l+i} = f^{l+1}\left[\sum_{j=1}^{N^l} w_{ij}^{l+1} f^l\left(\sum_{k=1}^{N^{l-1}} w_{jk}^l f^{l-1}\left(\ldots f^1\left(\sum_{n=1}^{N^0} w_{mn}^1 x_n + b_m^1\right)\ldots\right) + b_k^l\right) + b_j^{l+1}\right] \qquad [18]$$

2.4.2 Activation Functions

Each layer has an activation function that links the inputs to the outputs and therefore one layer to the next. There are various activation functions that are used in different situations and work better with different types of data. The activation functions can also be different among the hidden layers, moreover, special attention has to be put on the activation function of the output layer, because it determines whether the problem is regression or classification.

The activation functions have to have certain proprieties:

- be non-constant
- be bounded
- be monotonically increasing
- be continuous

In this project the functions used for the hidden layers are:

1. Sigmoid
2. ReLU (Rectified Linear Unit)
3. Leaky-RELU

and the one used for the output layer is:

4. Softmax

1. Sigmoid

$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad [19]$$

with derivative equal to:

$$\sigma'(z) = \sigma(z)\big(1-\sigma(z)\big) \qquad [20]$$

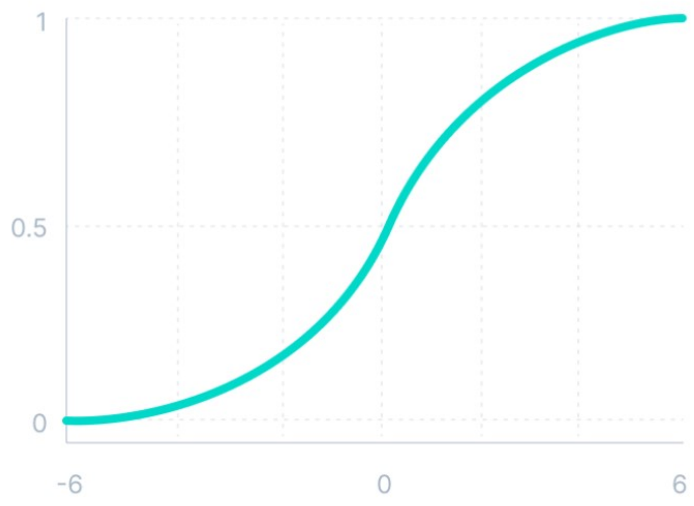


Figure 2: Sigmoid Function

The main reason why the sigmoid function is frequently used, is because it exists between 0 and 1. Therefore, it is especially used for models where we have to predict the probability as an output. However, when $|z|$ >> 0, then $\sigma'(z)$ is very small and it can lead to the "vanishing gradient" problem. This means that the gradient becomes zero, the training of the NN stops and the model is not able to find the right values for the weights. Therefore, other activation functions have been introduced to avoid this problem.

2. ReLU

$$ReLU(z) = \max\{0, z\} \tag{21}$$

with derivative equal to:

$$ReLU'(z) = \begin{cases} 1 & if\ z \geq 0 \\ 0 & if\ z < 0 \end{cases} \tag{22}$$

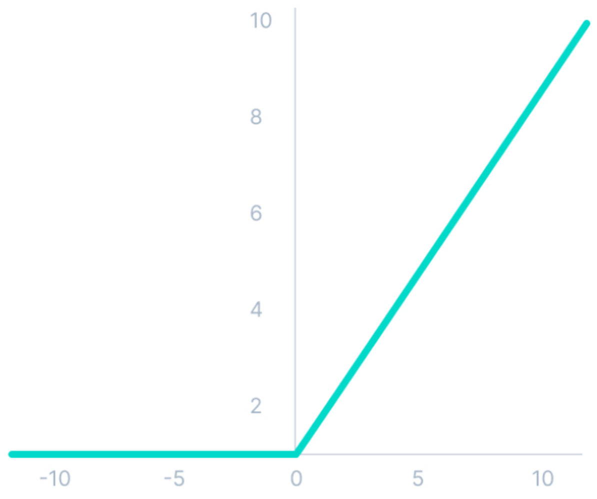


Figure 3: ReLU Function

The issue with the ReLU is that all the negative values become zero immediately and this decreases the ability of the model to properly train (and then fit) the data. To overcome this problem, the leaky-ReLU has been introduced.

3. Leaky-ReLU

$$L\_ReLU(z) = \begin{cases} z & if\ z \geq 0 \\ \alpha z & if\ z < 0 \end{cases} \qquad [23]$$

with derivative equal to:

$$L'_{ReLU}(z) = \begin{cases} 1 & if\ z \geq 0 \\ \alpha & if\ z < 0 \end{cases} \qquad [24]$$

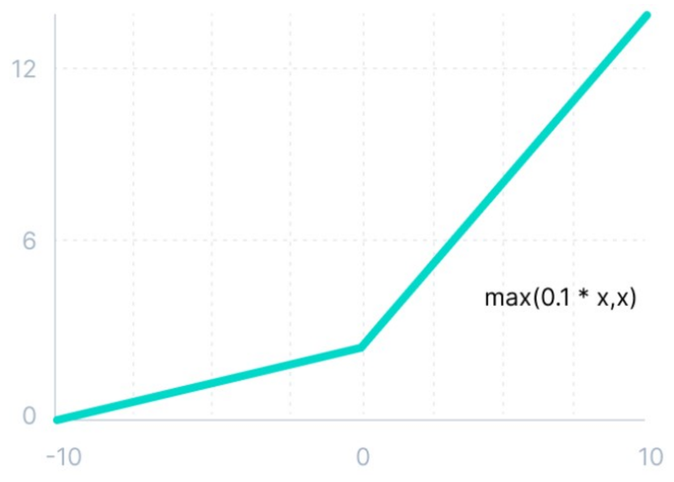


Figure 4: Leaky-ReLU Function

Usually, the value of α is 0.01. Using Leaky-ReLU the gradient never becomes zero. Empirically the Leaky-ReLU works better than the ReLU, but this may not always be the case.

4. Softmax

$$f(z_i) = \frac{e^{z_i}}{\sum_{m=1}^{K} e^{z_m}} \qquad [25]$$

with derivative with respect to z_i equal to:

$$f'(z_i) = f(z_i)\left(\delta_{ij} - f(z_j)\right) \qquad [26]$$

Softmax turns vectors into probabilities, which is exactly what the purpose of classification is. The exponential function in the formula ensures that the obtained values are non-negative. Due to the normalization term in the denominator, the obtained values sum to 1. Furthermore, all values lie between 0 and 1.

2.3.3 Back Propagation Algorithm

When training a FFNN the process starts with the random initialization of weights and biases, but in order to train properly the data, the weights need to be updated. To do so, the back propagation algorithm has been introduced. Let's define the cost function as:

$$C(\widehat{W}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - t_i)^2 \qquad [27]$$

where t_i are the targets and y_i are the outputs of the network.

Let's now define the activation of the j-th node of the l-th layer as function of the weights, the bias and the output of the previous layer:

$$z_j^l = \sum_{i=1}^{M^{l-1}} w_{ij}^l \, a_i^{l-1} + b_j^l \qquad [28]$$

Here M^{l-1} represents the total number of nodes of the $(l-1)$-th layer.

Starting from the activation values z^l in turn it is possible to define the output of layer l as:

$$a^l = f(z^l) \qquad [29]$$

From the definition of the activation z_j^l it can be written:

$$\frac{\partial z_j^l}{\partial w_{ij}^l} = a_i^{l-1} \qquad [30]$$

$$\frac{\partial z_j^l}{\partial a_i^{l-1}} = w_{ij}^l \qquad [31]$$

And then also:

$$\frac{\partial a_j^l}{\partial z_j^l} = a_j^l(1 - a_j^l) = f(z_j^l)\left(1 - f(z_j^l)\right) \qquad [32]$$

With these definitions it is possible to compute the derivative of the cost function in terms of the weights.

Let's refer to the output layer $l = L$. The cost function is:

$$C(\widehat{W^L}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - t_i)^2 = \frac{1}{2}\sum_{i=1}^{n}(a_i^L - t_i)^2 \qquad [33]$$

and the derivative of this function with respect to the weights is:

$$\frac{\partial C(\widehat{W^L})}{\partial w_{jk}^L} = (a_j^L - t_j)\frac{\partial a_{jk}^L}{\partial w_{jk}^L} \qquad [34]$$

The last partial derivative can be computed by applying the chain rule:

$$\frac{\partial a_j^L}{\partial w_{jk}^L} = \frac{\partial a_j^L}{\partial z_j^L}\frac{\partial z_j^L}{\partial w_{jk}^L} = a_j^L(1 - a_j^L)a_k^{L-1} \qquad [35]$$

Summing up, the first back propagation equation (the one referring to the output layer) is:

$$\frac{\partial C(\widehat{W^L})}{\partial w_{jk}^L} = (a_j^L - t_j)\frac{\partial a_{jk}^L}{\partial w_{jk}^L} = (a_j^L - t_j)a_j^L(1 - a_j^L)a_k^{L-1} \qquad [36]$$

and defining:

$$\delta_j^L = a_j^L(1 - a_j^L)(a_j^L - t_j) = f'(z_j^L)\frac{\partial C}{\partial a_j^L} \qquad [37]$$

Using the Hadamard product of two vectors this can be written as:

$$\delta^L = f'(z^L) \circ \frac{\partial C}{\partial a^L} \qquad [38]$$

In the Hadamard product of two vectors the elements corresponding to same row and columns of given matrices are multiplied together to form a new matrix.

The first term on the right-hand side measures how fast the activation function f is changing for a given activation value z_j^L. The second term on the right-hand side measures how fast the cost function is changing as function of the j-th output activation. If, for example, the cost function does not depend much on a particular output node j, then δ_j^L will be small.

By defining δ_j^L a more compact definition of the derivative of the cost function in terms of the weights can be written:

$$\frac{\partial C(\widehat{W^L})}{\partial w_{jk}^L} = \delta_j^L a_k^{L-1} \qquad [39]$$

Finally, generally speaking, the back propagation algorithm is:

1. Set up the input data x and initialize the weights and the biases

2. Perform a FFNN until the output layer is reached and compute all z^l and the pertinent outputs a^l for $l = 2,\ 3, \ldots,\ L$

3. Compute the error $\hat{\delta}^L$:

$$\hat{\delta}_j^L = f'(z_j^L)\frac{\partial C}{\partial a_j^L} \qquad [40]$$

4. Compute the back propagation error for each layer

$$\hat{\delta}_j^l = \sum_k \delta_k^{l+1}\, w_{kj}^{l+1}\, f'(z_j^l) \qquad [41]$$

5. Update the weights and biases using gradient descent for each layer

$$w_{jk}^l \leftarrow w_{jk}^l - \eta\ \delta_j^l\ a_k^{l-1} \qquad [42]$$

$$b_j^l \leftarrow b_j^l - \eta\ \delta_j^l \qquad [43]$$

The parameter η is the learning parameter discussed in connection with the gradient descent methods. It is convenient to use stochastic gradient descent with mini-batches. Once the back propagation algorithm reaches the first hidden layer, then a new FFNN starts, using the updated parameters. The process goes back and forth until the difference between the target and the outputs is small enough.

In the code I also added a regularization parameter lmb which has the purpose of constraining the size of the weights, so that they do not grow arbitrarily large to fit the training data, and in this way overfitting is reduced.

I will measure the size of the weights using the L2-norm, meaning the cost function becomes:

$$C(\Theta) = \frac{1}{N}\sum_{i=1}^{N} L_i(\Theta) \to \frac{1}{N}\sum_{i=1}^{N} L_i(\Theta) + \lambda||w||_2^2 = \frac{1}{N}\sum_{i=1}^{N} L_i(\Theta) + \lambda\sum_{ij} w_{ij}^2 \qquad [44]$$

Moreover, when dealing with a classification problem, the response vectors are transformed into matrices, following the one-hot vector representation. This means that each observation (each row of the matrix) gets as many values (columns) as the number of total classes. All the values will be 0 except for a unique value, which will be 1, and it will be found in the corresponding position of the class that each observation belongs to.

2.4 Decision Trees

Decision trees are supervised learning algorithms used for both classification and regression. The main idea of Decision trees is to find the features that contain the most information regarding the target, and then split the dataset along the values of these features such that the response values for the resulting underlying datasets are as pure as possible. The features which reproduce best the target are said to be the most informative ones. The process of finding the most informative features is done until a stopping criterion is accomplished.

A Decision tree has a root node, the interior nodes, and the final leaf nodes. These entities are then connected by so-called branches. The leaf nodes contain the information for the predictions because the model has learned the underlying structure of the training data and hence it can, given some assumptions, make predictions about the target feature value (class) of unseen observations.

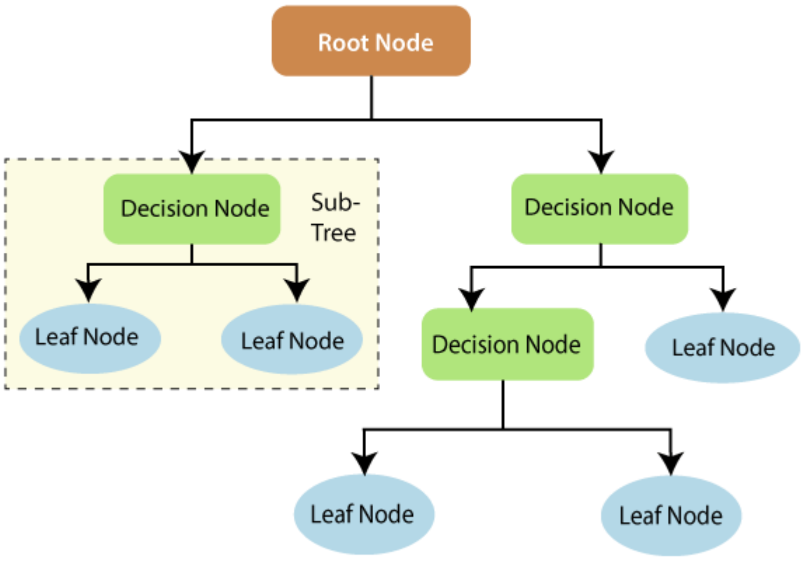


Figure 5: Decision Tree

The process of training a Decision tree and predicting the target of unseen observations is as follows:

1. Train the Decision tree model by continuously splitting the target feature along the values of the descriptive features using a measure of information gain during the training process
2. Grow the tree until we accomplish a stopping criterion, create leaf nodes which represent the predictions for new observations
3. Present the test set to the tree and run down the tree until the leaf nodes

Decision trees are used mainly because they are very easy to interpret. Moreover, they need no feature normalization. They can model nonlinear relationships and interactions between the different descriptive features. Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other classification approaches and they are prone to overfit the training data and hence do not well generalize the data if no stopping criteria or improvements like boosting or bagging are implemented. Small changes in the data may lead to a completely different tree (high variance), and this issue can be addressed by using Ensemble methods like Bagging, Boosting or Random forests.

2.4.1 Splitting Criteria

In this project the study is based on a classification problem, hence I am going to delve into this aspect of Decision trees. For a classification tree, the prediction belongs to the most commonly occurring class of training observations that are contained in the same region where that specific observation is. In order to grow a classification tree and evaluate the quality of a particular split, a recursive binary splitting is used: either the Gini index or the Entropy, since these two approaches are more sensitive to node purity than is the common classification error rate.

In order to split the branches of a tree, let's first define a PDF p_{mk} that represents the number of observations of a class k, with $k = 1,2,\dots,K$, in a region R_m with N_m observations.This likelihood function is represented in terms of the proportion $I(y_i = k)$ of observations of this class in the region R_m as:

$$p_{mk} = \frac{1}{N} \sum_{i \in R_m} I\,(y_i = k) \qquad [45]$$

With p_{mk} representing the majority class of observations in region m. The two ways of splitting a node that have been implemented in this project are:

1. **Gini index g:**

$$g = \sum_{k \neq k'} p_{mk}\, p_{mk'} = \sum_{k=1}^{K} p_{mk}\,(1 - p_{mk}) \qquad [46]$$

The Gini index g gives the degree of probability of a specific variable that is wrongly classified. It takes values $g \in [0,1]$ with $g = 0$ if all elements belong to one class only and $g = 1$ if all elements are randomly distributed across various classes. A value $g = 0.5$ means that the elements in a node are uniformly distributed across classes.

2. **Entropy s**:

$$s = -\sum_{k=1}^{K} p_{mk}\, log(p_{mk}) \qquad [47]$$

Entropy is a measure of uncertainty, and the goal is to reduce it. It takes values $s \in [0,1]$ and it is lowest at the extremes, when all the elements are all the same, which means that the region is pure and the disorder is 0. Entropy is highest in the middle, for $p = 0.5$ (in the case of just two classes), when the elements are evenly split between different classes. That is extreme disorder, because there is no majority.

2.4.2 Algorithm

In this project I have used Python's library Scikit-learn for training and predicting Decision trees, which implements the CART algorithm (Classification And Regression Tree).

The CART algorithm splits the data set in two subsets using a single feature k and a threshold t_k. It searches for the pair (k,t_k) that produces the purest subset using either the Gini factor G or the Entropy. Assuming it uses the Gini criterion, the cost function it tries to minimize is:

$$C(k,t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right} \qquad [48]$$

where G_{left} and G_{right} measure the impurity of the left/right subset and m_{left} and m_{right} are the number of instances in the left/right subset.

Once it has successfully split the training set in two, it splits the two subsets using the same logic, then the subsubsets and so on, recursively. It stops once it reaches the maximum depth (defined by the max_depth hyperparameter), or if it cannot find a split that will reduce impurity.

2.5 Ensemble Methods

As already mentioned, Decision trees can often end up overfitting the training data, which means that the model has a high variance. In order to reduce it, the Ensemble methods have been introduced. These methods combine several base models in order to produce one optimal predictive model and they allow the researchers to take into account a sample of i.e.Decision trees, calculate which features to use at each split and make a final predictor based on the aggregated results of the sampled Decision trees. The base learner does not have to be a decision tree, any estimator could be used, but it should be weak, so that the aggregation of multiple weak estimators could lead to a strong estimator. A weak classifier has to be intended as a classifier which produces results that are only slightly better than those acquired via a random guess.

2.6 Bagging

The plain Decision trees suffer, indeed, from high variance. This means that, if training data was to be split into two parts at random, and fit a Decision tree to both halves, the results could be quite different. In contrast, a procedure with low variance will yield similar results if applied repeatedly to distinct datasets.

Bootstrap Aggregation, or just Bagging, is an ensemble method that reduces the variance of a statistical learning method, and in this project it is used with decision trees. Bagging typically results in improved accuracy over prediction using a single tree. Unfortunately, however, the resulting model loses the simplicity of interpretation, which is the biggest advantage of Decision trees. In fact, when bagging a large number of trees, it is no longer possible to represent the resulting statistical learning procedure using a single tree, and it is no longer clear which variables are most important to the procedure. Hence, Bagging improves prediction accuracy at the expense of interpretability.

Let's assume there's a sample dataset of M observations and that it has been implemented a Decision tree on it. The algorithm for Bagging for a classification problem is:

1. Create many random sub-samples of the dataset with replacement

2. Train a Decision tree on each sample

3. Given a new dataset, calculate the average prediction from each model and choose the final prediction based on the class that has the highest frequency.

2.7 Random Forests

Random forests provide an improvement over bagged trees by way of a small change that decorrelates the trees. As in Bagging, Random forests build a number of Decision trees on bootstrapped training samples, but when building these Decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those $m < p$ predictors. A new sample of m predictors is taken at each split, and typically it is chosen to be $m \approx \sqrt{p}$.

Let's suppose that there is one very strong predictor in the data set, then in the collection of bagged trees, most or all the trees will use this strong predictor in the top split. Consequently, all the bagged trees will look quite similar to each other. Hence the predictions from the bagged trees will be highly correlated. Unfortunately, averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities. In particular, this means that bagging will not lead to a substantial reduction in variance over a single tree. On the other hand, in Random forests, each split is determined based only on $m < p$ features, which leads to building Decision trees quite different from each other, which means that they are less correlated, hence the resulting model has a bigger reduction in variance.

2.7.1 Algorithm

Let's suppose the forest has B trees, then for each $b = 1,2,\dots,B$:

1. Draw a bootstrap sample from the training data

2. Grow a Random forest tree T_b based on the bootstrapped data by repeating these following steps until the maximum node size is reached:

    2a. Select $m < p$ variables at random from the p features

    2b. Pick the best split point among the m features (i.e. using CART) and create a new node

    2c. Split the node into daughter nodes

3. The output is the ensemble of trees given by $\{T_b\}_1^B$ and it is ready to make predictions by fitting the test set.

2.8 Boosting

The main idea of boosting is the same as the other Ensemble methods, namely reducing the variance and therefore improving the prediction ability by aggregating multiple predictors, but boosting does so through an iterative path that emphasizes those observations which have been misclassified in the previous step, by weighting them with a bigger factor.

2.8.1 Adaptive Boosting: AdaBoost

Let us consider the binary classification problem of this project with two outcomes $y_i \in \{-1,1\}$ and $i = 0,1,\dots,n-1$ as set of observations. Let's define a classification function G(x) which produces a prediction taking one of the two values $\{-1,1\}$. The error rate of the training sample is then:

$$\overline{err} = \frac{1}{n}\sum_{i=0}^{n-1} \mathrm{I}\big(y_i \neq G(x_i)\big) \qquad [49]$$

The iterative procedure starts with defining a weak classifier whose error rate is barely better than random guessing. The iterative procedure in Boosting is to sequentially apply a weak classification algorithm to repeatedly modified versions of the data producing a sequence of weak classifiers $G_m(x)$.

Let's express the function f(x) in terms of G(x). That is:

$$f_M(x) = \sum_{i=1}^{M} \beta_m \, b(x; \gamma_m) \qquad [50]$$

that is a function of:

$$G_M(x) = sign \sum_{i=1}^{M} \alpha_m \, G_m(x) \qquad [51]$$

In the iterative procedure:

$$f_m(x) = f_{m-1}(x) + \beta_m G_m(x) \qquad [52]$$

The function at the m-th step is determined by the one at $(m-1)$-th step and the value of G at the m-th step.

The simplest possible cost function which leads to the AdaBoost algorithm is the exponential cost function defined as:

$$C(y, f) = \sum_{i=0}^{n-1} exp\Big(-y_i\big(f_{m-1}(x_i) + \beta G(x_i)\big)\Big) \qquad [53]$$

β and G need to be optimized for each value of $m = 1, \dots, M$ and it is usually done in two steps.

Let's, however, first rewrite the cost function as:

$$C(y,f) = \sum_{i=0}^{n-1} w_i^m \, exp\big(-y_i \beta G(x_i)\big) \qquad [54]$$

where I have defined:

$$w_i^m = exp\big(-y_i f_{m-1}(x_i)\big) \qquad [55]$$

in order to get to the AdaBoost algorithm, G needs to be optimized for any $\beta > 0$:

$$G_m(x) = sign \sum_{i=0}^{n-1} I\,(y_i \neq G\,(x_i)) w_i^m \qquad [56]$$

which is the classifier that minimizes the weighted error rate in predicting y.

Let's rewrite this as:

$$\text{exp} -(\beta) \textstyle\sum y_i = G(x_i) w_i^m + exp(\beta) \sum y_i \neq G(x_i) w_i^m \qquad [57]$$

which can be rewritten again as:

$$(exp(\beta) - ex\,p - (\beta)) \sum_{i=0}^{n-1} I\,\big(y_i \neq G(x_i)\big) w_i^m + exp(-\beta) \sum_{i=0}^{n-1} w_i^m = 0 \qquad [57]$$

which leads to:

$$\beta_m = \frac{1}{2} log \frac{1 - \overline{err}}{\overline{err}} [58]$$

where the error has been redefined as:

$$\overline{err_m} = \frac{1}{n} \frac{\sum_{i=0}^{n-1} w_i^m \, \mathrm{I}\big(y_i \neq G\,(x_i)\big)}{\sum_{i=0}^{n-1} w_i^m} \qquad [59]$$

which leads to an update of:

$$f_m(x) = f_{m-1}(x) + \beta_m G_m(x) \qquad [60]$$

This leads to the new weights:

$$w_i^{m+1} = w_i^m \exp\left(-y_i \beta_m G_m(x_i)\right) \tag{61}$$

2.8.1.1 Algorithm

Let's assume that the weak classifier is a decision tree and let's consider a binary set of outputs with $y_i \in \{-1,1\}$ with $i = 0,1,\ldots,n-1$ as set of observations. Let's also define a classifier determined by the data via a function G(x). This function explains how well the classification for the target y is.

Let's recall the misclassification error err: [49]

$$err = \frac{1}{n}\sum_{i=0}^{n-1} I\left(y_i \neq G(x_i)\right)$$

where the function $I(y_i \neq G(x_i)) = 1$ if the i-th observation is misclassified and $I(y_i \neq G(x_i)) = 0$ if it is correctly classified.

Then let's rewrite the misclassification error as:

$$\overline{err_m} = \sum_{i=0}^{n-1} w_i^m \, \mathrm{I}\left(y_i \neq G(x_i)\right) \sum_{i=0}^{n-1} w_i \tag{62}$$

The idea is to set up weights which will be used to scale the correctly classified and the misclassified cases.

1. Initialize all weights to $w_i = \frac{1}{n}$, with $i = 0,1,\ldots,n-1$

2. For each $m = 1,2,\ldots,M$ repetitions:

    2a. Fit a given classifier to the training set using the weights w_i

    2b. Compute err_m and figure out which events are classified properly and which are classified wrongly

2c. Define the quantity:

$$\alpha_m = \frac{\log(1 - \overline{err_m})}{\overline{err_m}} \quad [63]$$

2d. Set the new weights to:

$$w_i = w_i exp\ (\alpha_m I(y_i \neq G(x_i))) \quad [64]$$

3. Compute the new classifier:

$$G(\mathrm{x}) = \sum_{i=0}^{n-1} \alpha_m\ I(y_i \neq G(x_i))) \quad [65]$$

The observations which were misclassified at iteration $(m - 1)$ have a weight which is larger than those which were classified properly. As this algorithm proceeds, the observations which were difficult to classify correctly are given a larger influence. Each new classification step m is forced to concentrate on those observations that are missed in the previous iterations.

2.8.2 Gradient Boosting

Gradient boosting is a similar technique to Adaptive Boosting, it combines the so-called weak classifiers into a strong method via a series of iterations.

2.8.2.1 Algorithm

Let's suppose the cost function is given by $C(f) = \sum_{i=0}^{n-1} L\ (y_i, f(x_i))$ where y_i is the target and $f(x_i)$ is the function which is meant to model y_i, then:

1. Initialize the estimate $f_0(x)$

2. For $m = 1,2,\ldots M$:

2a. Compute the negative gradient vector:

$$u_m = -\frac{\partial C(y,f)}{\partial f\ (x)} \quad [66]$$

at $f(x) = f_{m-1}(x)$

2b. Fit the base learner to the negative gradient:

$$h_m(u_m, x) \tag{67}$$

2c. Update the estimate:

$$f_m(x) = f_{m-1}(x) + h_m(u_m, x) \tag{68}$$

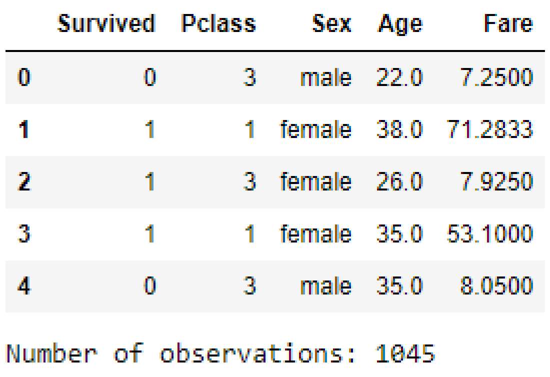
3. The final estimate is then:

$$f_M(x) = \sum_{m=1}^{M} h_m(u_m, x) \tag{69}$$

3. Analysis

3.1 Descriptive Analysis

Let’s take a look at the dataset I am going to work with:

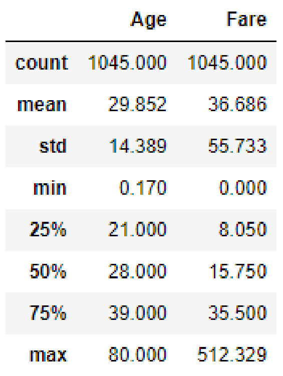


	Survived	Pclass	Sex	Age	Fare
0	0	3	male	22.0	7.2500
1	1	1	female	38.0	71.2833
2	1	3	female	26.0	7.9250
3	1	1	female	35.0	53.1000
4	0	3	male	35.0	8.0500

```
Number of observations: 1045
```

Table 1: Dataset

It is essential to understand how the data is distributed and if there are specific patterns among the observations. To do so, I am going to calculate some descriptive univariate analysis both on the continuous features and on the categorical features.



	Age	Fare
count	1045.000	1045.000
mean	29.852	36.686
std	14.389	55.733
min	0.170	0.000
25%	21.000	8.050
50%	28.000	15.750
75%	39.000	35.500
max	80.000	512.329

Table 3: Descriptive Analysis for Continuous Features

The average *Age* of the passengers is 30 years old and the average price they paid to get on the Titanic is approximately 36.7$. However, the variable *Fare* presents a way bigger variance than the variable *Age* (std = 55.7 vs std = 14.3), this means that the range of the prices paid by the passengers is way bigger than the range of their age. Indeed, *Fare* goes from 0$ (crew) to 512.3$. It is noticeable how wide it is compared to the *Age* (0.2; 80) years-old. From the table, it is clear that the 80-year-old passenger is probably one of the few elder people on the Titanic, since the third quartile can be found at 39 years-old. This same aspect is even more evident for that variable *Fare*, since the third quartile is 35.5$ and the maximum values is 512.3$. This means that most of the people paid very little for their ticket and just a few of them spent a very high amount of money. This means that both these distributions, and especially the one of *Fare* are asymmetric with a long and heavy right tail.

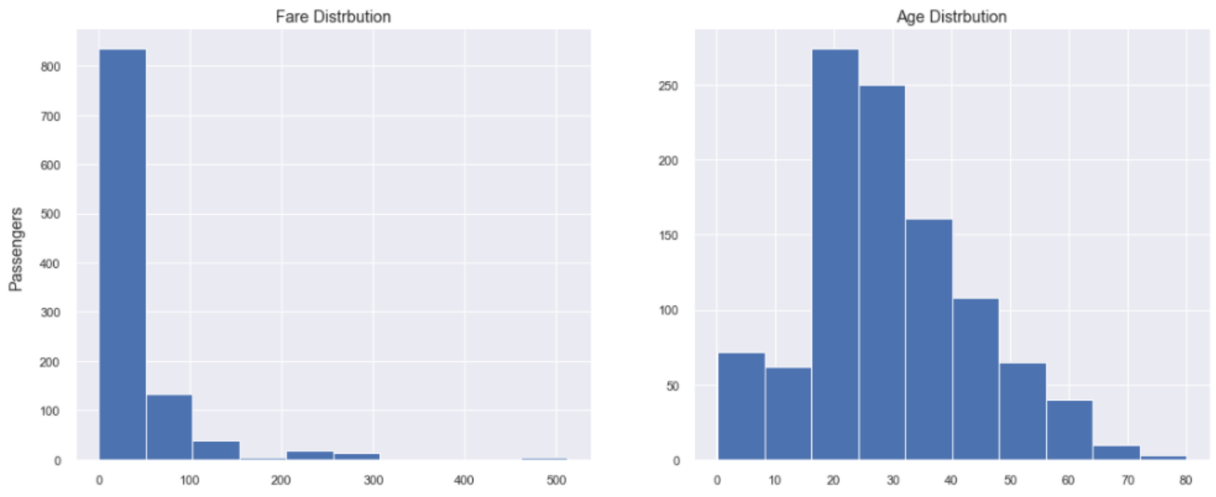


Figure 6: Fare and Age Distribution

There are 500 people in third class, 261 in second class and 284 in first class, and it is noticeable how the mean age increases as they go from third to first class. Obviously, also the mean fare increases, going from an average of 12.9$ for the third class to 82.6$ for the first class.

	Age								Fare							
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	max
Sex																
female	388.0	28.687088	14.576962	0.17	19.0	27.0	38.0	76.0	388.0	50.308775	67.344745	6.75	13.0000	26.0	59.400	512.3292
male	657.0	30.539696	14.243582	0.33	21.0	28.0	39.0	80.0	657.0	28.641019	45.750689	0.00	7.8958	13.0	29.125	512.3292

Table 4: Descriptive Analysis grouped by *Pclass*

There are 388 females and almost the double (657) males. The males are, on average, slightly older than the women, but there is not a relevant difference between the two distributions. When it comes to the price of the ticket, females spend on average almost twice as much as the men do: 50.3$ versus 28.6$. This also because females are not part of the crew, whereas men are. Also the quartiles, and not only the minimum, show higher values for women then for men. The highest price for the ticket, however, is the same for men and women, probably was a couple or someone travelling together.

	Age								Fare							
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	max
Survived																
0	628.0	30.463232	13.871620	0.33	21.0	28.0	39.0	74.0	628.0	25.798917	36.804437	0.0	7.8958	13.0	27.7208	263.0000
1	417.0	28.931079	15.105581	0.17	19.0	28.0	38.0	80.0	417.0	53.082096	72.836292	0.0	13.0000	26.0	63.3583	512.3292

Table 5: Descriptive Analysis grouped by *Sex*

Let's take a look at the response variable. There are 628 that did not survive the sinking, and 417 who did.

	Age								Fare							
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	max
Pclass																
1	284.0	39.159930	14.548028	0.92	28.0	39.0	50.00	80.0	284.0	92.229358	82.601201	0.0000	35.28855	69.30	110.88330	512.3292
2	261.0	29.506705	13.638627	0.67	22.0	29.0	36.00	70.0	261.0	21.855044	13.540335	9.6875	13.00000	15.75	26.00000	73.5000
3	500.0	24.745000	11.862896	0.17	18.0	24.0	31.25	74.0	500.0	12.879299	9.733090	0.0000	7.77500	8.05	15.13645	69.5500

Table 6: Descriptive Analysis grouped by *Survived*

Between them there is not a significant difference in *Age,* suggesting that the age of a passenger did not heavily influence the outcome. Taking a look at the average fare prices of those who survived, it is clear that it is higher than the one paid by those who did not survive. However, also people from the first class did not survive, being the maximum 263$.

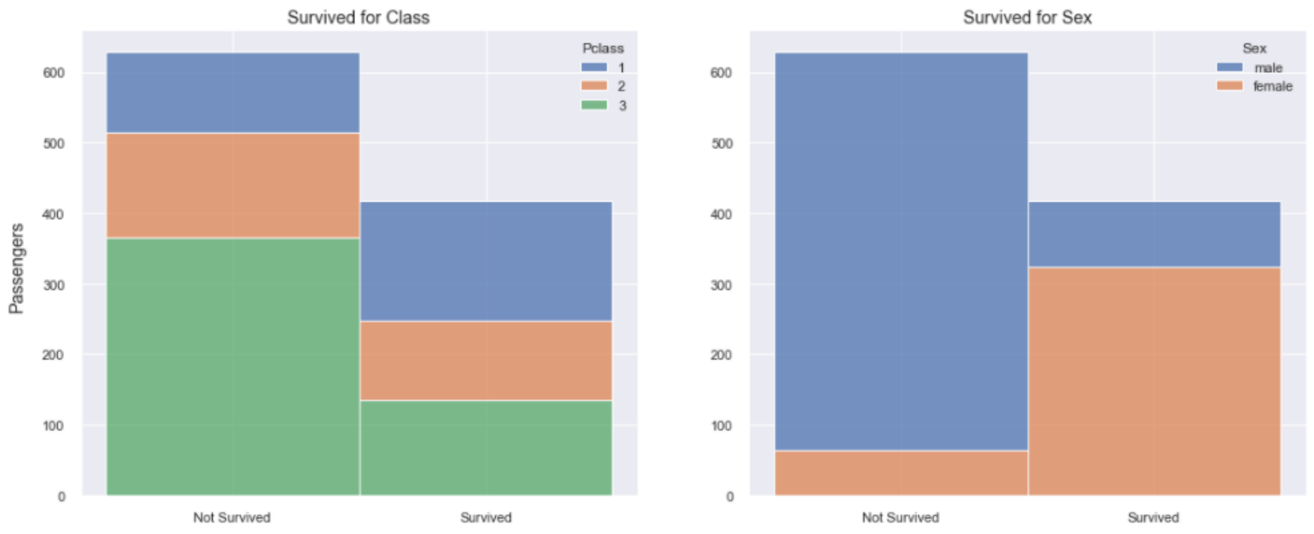


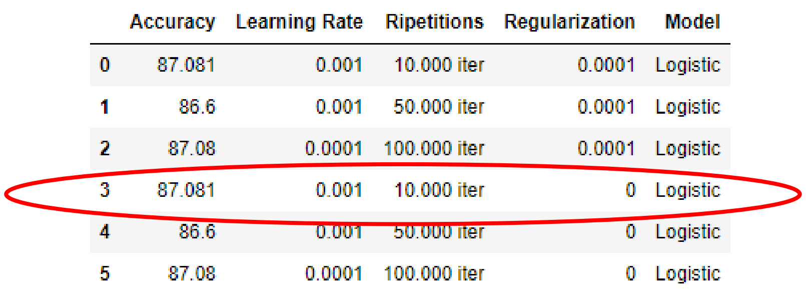
Figure 7: Distribution of *Survived* grouped by *Pclass* and *Sex*

As already observed in Table 4, among those who did not survive there are more passengers in third class and fewer form first class. Analogously, among those who survived there are more passengers from first class and fewer form third class. However, the big difference is noticeable in the graph on the right: very little women did not survive (16.5%) and very little men did (14.1%).

3.2 Logistic Regression

The project aims at analyzing the prediction capacity of Regression, Neural Networks and Ensemble Methods. In this first section I will implement a Logistic Regression both with and without regularization, each time calculating the *Accuracy Score,* with the goal of maximizing it. This process has been repeated 9 times. Initially it has been implemented with intercept and L2 regularization, running it every time with a different number of iterations and with a grid search to find the optimal learning rate.

Secondly without fitting the intercept and it resulted that for each number of iterations, there was no difference between adding or keeping out the intercept. Lastly, I ran the algorithm without any penalization of the parameters and once again, the results were identical to the ones obtained with regularization. It is noticeable that even when included, the values of the regularization parameter were very close to 0.



	Accuracy	Learning Rate	Ripetitions	Regularization	Model
0	87.081	0.001	10.000 iter	0.0001	Logistic
1	86.6	0.001	50.000 iter	0.0001	Logistic
2	87.08	0.0001	100.000 iter	0.0001	Logistic
3	87.081	0.001	10.000 iter	0	Logistic
4	86.6	0.001	50.000 iter	0	Logistic
5	87.08	0.0001	100.000 iter	0	Logistic

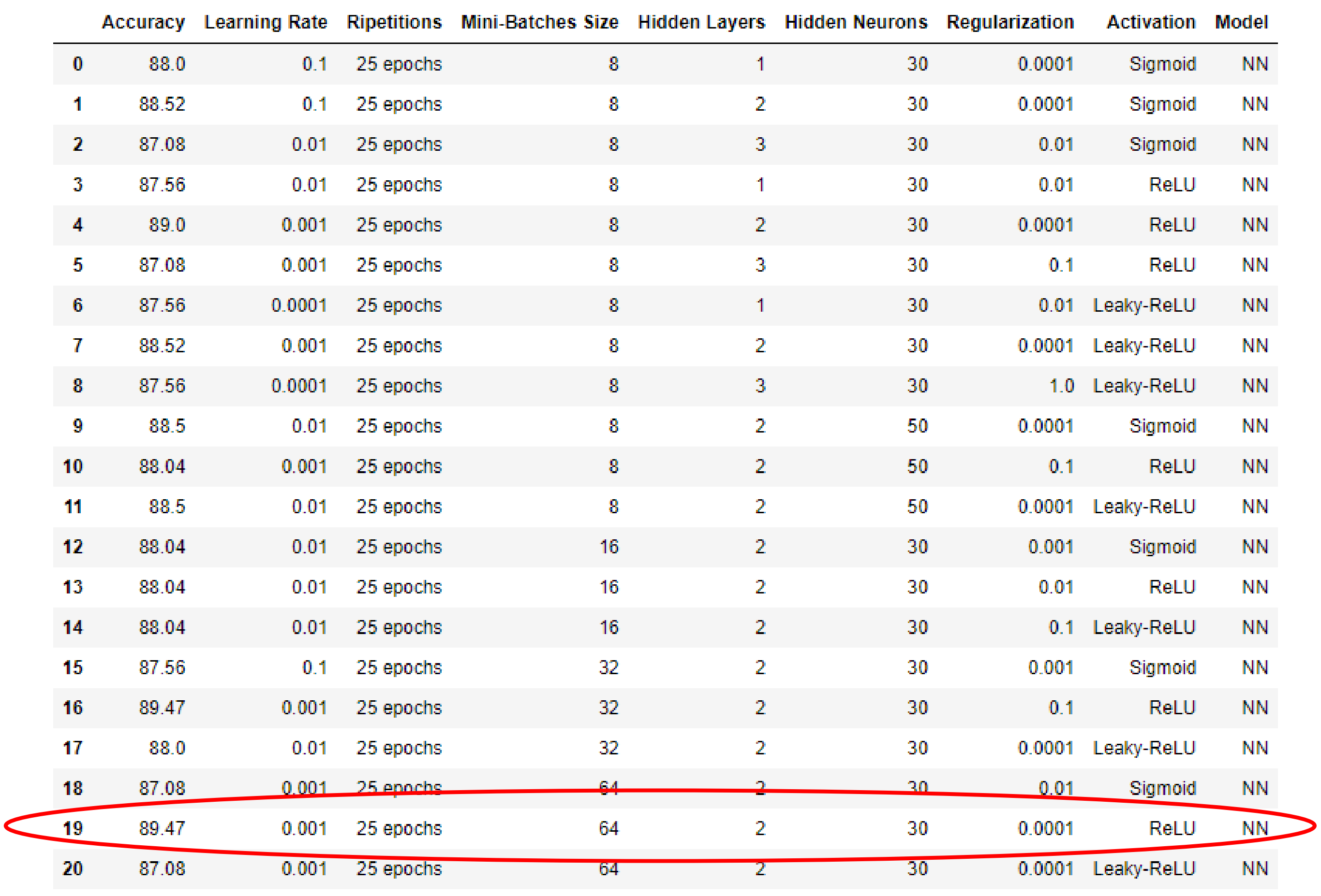
Table 7: Accuracy Score for Logistic Regression

It appears that for this dataset the best logistic regression model can avoid tuning λ as hyperparameter, however, it is often proved that a penalization parameter actually improves the goodness of prediction.

3.3 Neural Networks

In order for the model to be able to classify the observations into two categories, the final output of the MLP has be a probability, which will be approximated to 0 if it is ≤ 0.5 and to 1 if it is >0.5. To obtain a probability as last output a_j^L, the chosen activation function is the softmax.

For this classification problem, the activation function ReLU works best than the sigmoid and Leaky-ReLU functions, which turned out to be the best option for regression. However, as already mentioned, this cannot be considered a ready-made rule. After training the first model, it was evident that the model with 2 hidden layers were the ones producing better predictions, so I implemented the following model keeping this setting and changing the size of the mini-batches, the number of hidden nodes and the activation functions. Moreover, both the optimal learning rate and regularization parameter were found through a search grid. The best model among the ones trained is a model trained with ReLU as activation function and has an accuracy of 89,47%.



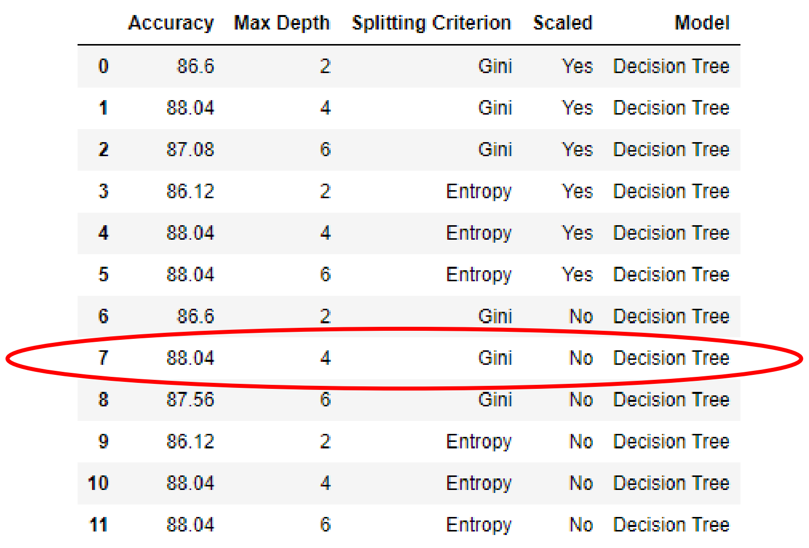
	Accuracy	Learning Rate	Ripetitions	Mini-Batches Size	Hidden Layers	Hidden Neurons	Regularization	Activation	Model
0	88.0	0.1	25 epochs	8	1	30	0.0001	Sigmoid	NN
1	88.52	0.1	25 epochs	8	2	30	0.0001	Sigmoid	NN
2	87.08	0.01	25 epochs	8	3	30	0.01	Sigmoid	NN
3	87.56	0.01	25 epochs	8	1	30	0.01	ReLU	NN
4	89.0	0.001	25 epochs	8	2	30	0.0001	ReLU	NN
5	87.08	0.001	25 epochs	8	3	30	0.1	ReLU	NN
6	87.56	0.0001	25 epochs	8	1	30	0.01	Leaky-ReLU	NN
7	88.52	0.001	25 epochs	8	2	30	0.0001	Leaky-ReLU	NN
8	87.56	0.0001	25 epochs	8	3	30	1.0	Leaky-ReLU	NN
9	88.5	0.01	25 epochs	8	2	50	0.0001	Sigmoid	NN
10	88.04	0.001	25 epochs	8	2	50	0.1	ReLU	NN
11	88.5	0.01	25 epochs	8	2	50	0.0001	Leaky-ReLU	NN
12	88.04	0.01	25 epochs	16	2	30	0.001	Sigmoid	NN
13	88.04	0.01	25 epochs	16	2	30	0.01	ReLU	NN
14	88.04	0.01	25 epochs	16	2	30	0.1	Leaky-ReLU	NN
15	87.56	0.1	25 epochs	32	2	30	0.001	Sigmoid	NN
16	89.47	0.001	25 epochs	32	2	30	0.1	ReLU	NN
17	88.0	0.01	25 epochs	32	2	30	0.0001	Leaky-ReLU	NN
18	87.08	0.001	25 epochs	64	2	30	0.01	Sigmoid	NN
19	89.47	0.001	25 epochs	64	2	30	0.0001	ReLU	NN
20	87.08	0.001	25 epochs	64	2	30	0.0001	Leaky-ReLU	NN

Table 8: Accuracy Score for Neural Networks

There are two different models that show this accuracy score, both trained on ReLU, one with mini-batch size of 32 and the other of 64. Since it is always better to choose the most computationally convenient model, the best will be the one with more observation in the mini-batch, because it means there are less mini-batches to go through. The optimal learning rate is small (0.001), which means that the gradient descent moved quite slowly towards the maximum. The regularization parameter, however, is very small (0.0001), which means that the weights are not heavily penalized, which brings the algorithm to converge slower to the maximum, probably because it is not that far away from the initial value.

3.4 Decision Trees

Decision trees do not require the data to be scaled in order to be fitted, but as mentioned earlier, the scaled data present a smaller variance. Let's see if using scaled data actually improves the prediction ability of the tree.



	Accuracy	Max Depth	Splitting Criterion	Scaled	Model
0	86.6	2	Gini	Yes	Decision Tree
1	88.04	4	Gini	Yes	Decision Tree
2	87.08	6	Gini	Yes	Decision Tree
3	86.12	2	Entropy	Yes	Decision Tree
4	88.04	4	Entropy	Yes	Decision Tree
5	88.04	6	Entropy	Yes	Decision Tree
6	86.6	2	Gini	No	Decision Tree
7	88.04	4	Gini	No	Decision Tree
8	87.56	6	Gini	No	Decision Tree
9	86.12	2	Entropy	No	Decision Tree
10	88.04	4	Entropy	No	Decision Tree
11	88.04	6	Entropy	No	Decision Tree

Table 9: Accuracy Score for Decision Trees

As noticeable, there is no difference between the accuracy between scaled and unscaled data. Since, however, the unscaled data permits an easy interpretation, it is preferable. I have also implemented both splitting criteria, and in this case, there is little to none difference between the two. What makes the difference is the depth of the tree: how many branches does a tree have. Six branches are often empirically considered the optimal depth, but in this case produces often as good results as four branches, if not worse. The best model is given indeed by 4 branches and unscaled data, with either criterion (Gini or Entropy). It is preferable to choose a smaller tree if they have the same accuracy.

As mentioned, Decision trees' best advantage is its interpretability, in fact by plotting the outcome it is possible to easily classify all the observations. From Figure 8 it is deducible that the most informative variable is *Sex*, since it is the one used for the first split. Based just on this first split the classification is: Male = Not Survived, Female = Survived. Which was expected from Figure 7.

Adding the second split the predictions become more accurate: Male under the age of twelve = Not Survived, Male over the age of twelve = Survived, Female in second or first class = Survived, Female in third class = Not Survived. And so on. The best model among the ones trained has a depth of four branches, which means that four level of splits are enough to make good enough predictions.

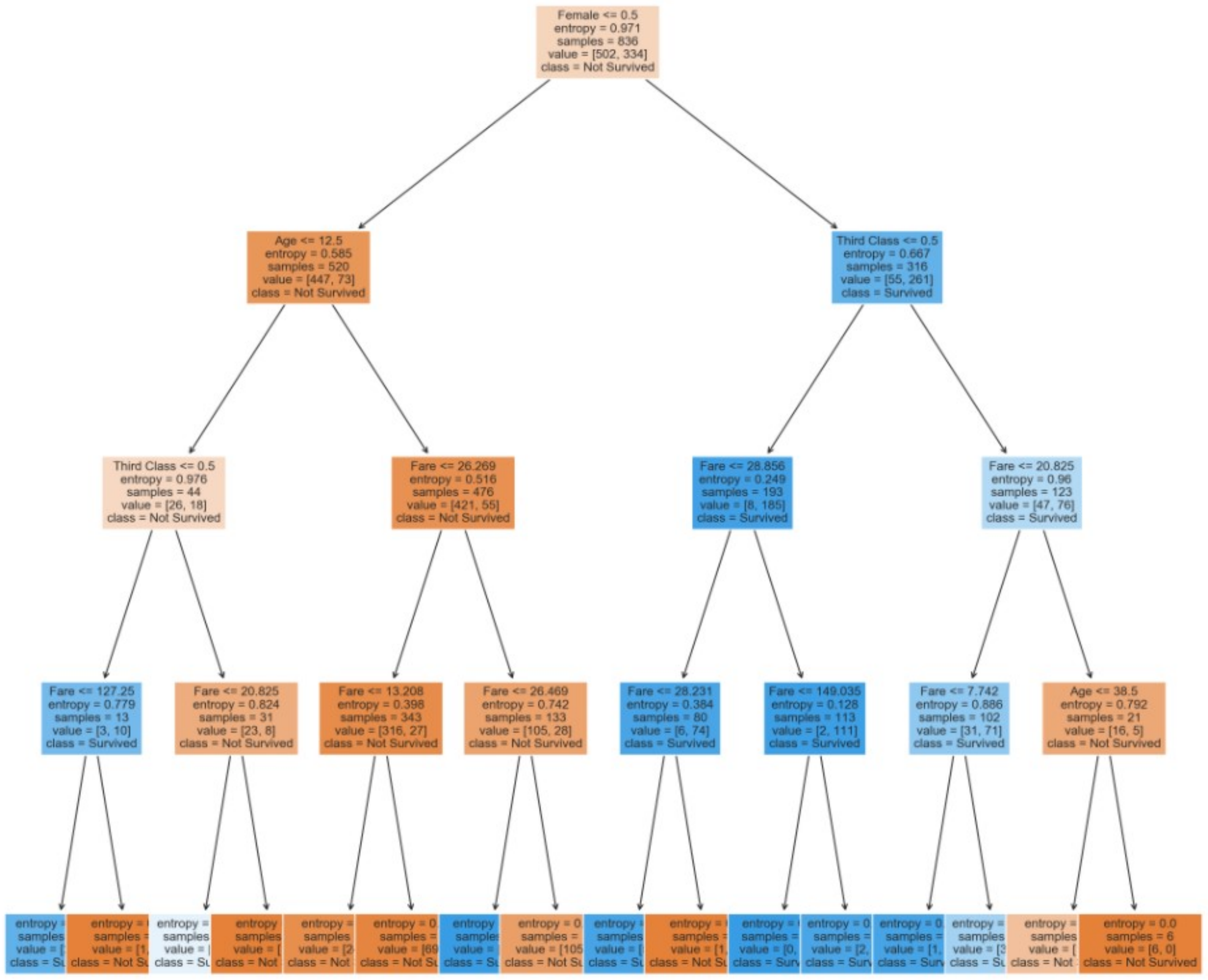
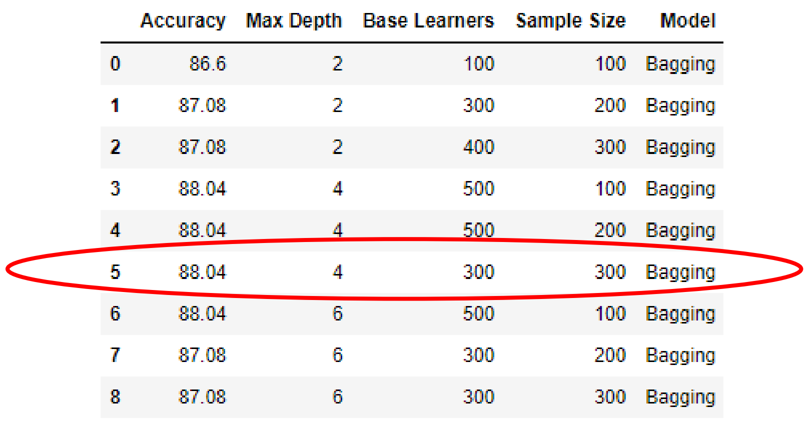


Figure 8: Best Decision Tree

3.5 Bagging

The trees used for this Bagging models have been trained with 2, 4 and 6 branches and the number of trees used (base learners) goes from 100 to 500, with sample size of each base learner of 100, 200 and 300. The model trained on trees with 4 branches produces the same accuracy score of some of the models trained on tree with 6 branches, but better than others. This means that probably in some models 6 branches are too many and the splits become not relevant.



	Accuracy	Max Depth	Base Learners	Sample Size	Model
0	86.6	2	100	100	Bagging
1	87.08	2	300	200	Bagging
2	87.08	2	400	300	Bagging
3	88.04	4	500	100	Bagging
4	88.04	4	500	200	Bagging
5	88.04	4	300	300	Bagging
6	88.04	6	500	100	Bagging
7	87.08	6	300	200	Bagging
8	87.08	6	300	300	Bagging

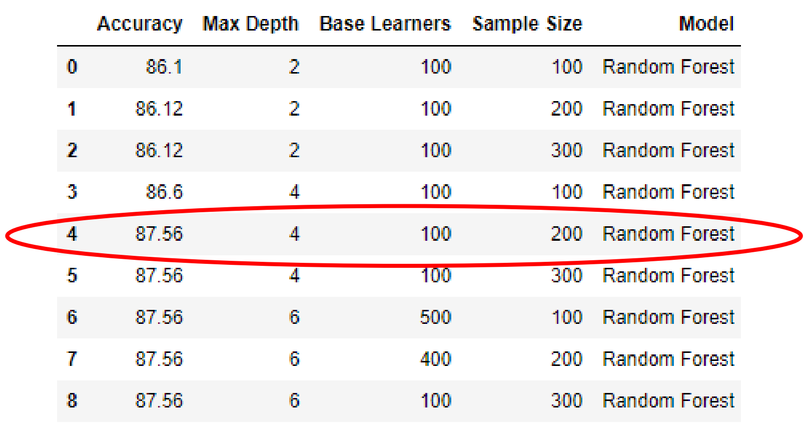
Table 9: Accuracy Score for Bagging

For each combination of depth and sample size the table shows the number of base learners that produces the best accuracy score

The best model is given by 300 trees with 4 branches with 300 observations each. This accuracy is exactly the same obtained with the simple Decision Tree (88.04%). This happens because the bagging is not always able to reduce the variance of a decision tree if all the bagged decision trees are very similar to each other, because of high correlation between them. Since with bagging the interpretability of the Decision Tree is lost, it could be preferable in this case to use the single tree as best model between the two.

3.6 Random Forests

For Random forests I implemented the algorithm with the same setting as in Bagging. This means that the trees have been trained with 2, 4 and 6 branches and the number of trees used goes from 100 to 500, with sample size of each base learner of 100, 200 and 300. Once again, the model trained on trees with 4 branches produces the highest accuracy and it happens again that it has the same accuracy score of the models trained on tree with 6 branches. This means that probably in some models 6 branches are too many and the splits become not relevant. The best model is given by 100 trees with 4 branches with 200 observations each.



	Accuracy	Max Depth	Base Learners	Sample Size	Model
0	86.1	2	100	100	Random Forest
1	86.12	2	100	200	Random Forest
2	86.12	2	100	300	Random Forest
3	86.6	4	100	100	Random Forest
4	87.56	4	100	200	Random Forest
5	87.56	4	100	300	Random Forest
6	87.56	6	500	100	Random Forest
7	87.56	6	400	200	Random Forest
8	87.56	6	100	300	Random Forest

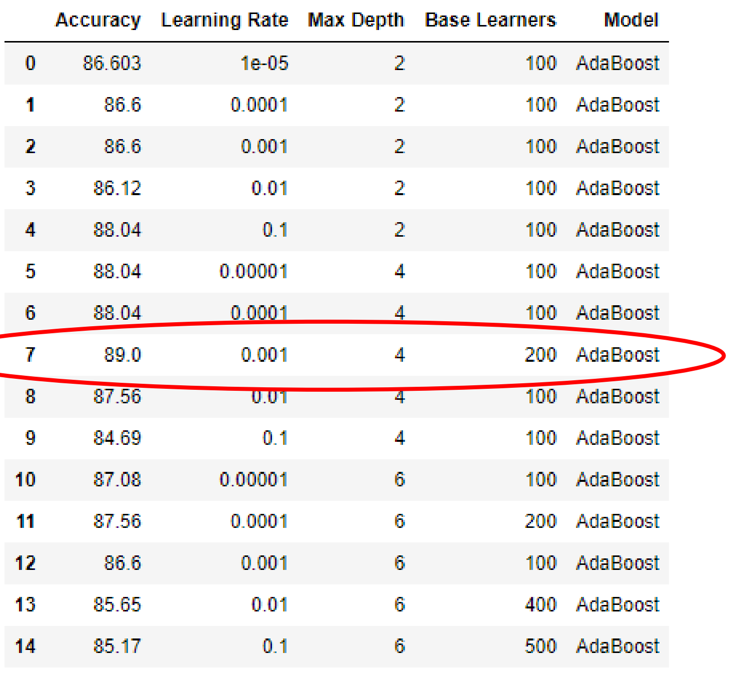
Table 10: Accuracy Score for Random Forests

For each combination of depth and sample size the table shows the number of base learners that produces the best accuracy score

Even though Random forests are supposed to work better than single trees, in this project the Decision tree produces a better accuracy (88.04% versus 87.56%). In Random forests, at each split the algorithm chooses the split variable among $m \approx \sqrt{p} \approx \sqrt{7} \approx 2$ features. This means that it only has two options, and if those two are not relevant variables, then that split is not significant, and it does not contribute positively to good prediction making. If there were more features, than Random forests would work better because

3.7 AdaBoost

The trees used for these AdaBoost models have been trained with 2, 4 and 6 branches and the number of trees used goes from 100 to 500, with learning rate from 0.00001 to 0.1. The best model has once again been trained on a tree with 4 branches of depth, moreover it has 200 base learners and a learning rate of 0.001.



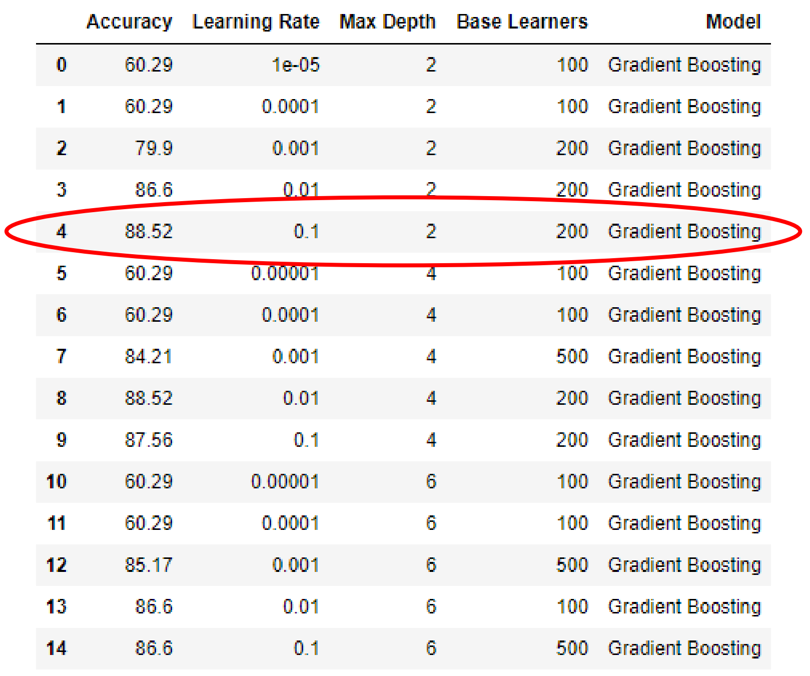
	Accuracy	Learning Rate	Max Depth	Base Learners	Model
0	86.603	1e-05	2	100	AdaBoost
1	86.6	0.0001	2	100	AdaBoost
2	86.6	0.001	2	100	AdaBoost
3	86.12	0.01	2	100	AdaBoost
4	88.04	0.1	2	100	AdaBoost
5	88.04	0.00001	4	100	AdaBoost
6	88.04	0.0001	4	100	AdaBoost
7	89.0	0.001	4	200	AdaBoost
8	87.56	0.01	4	100	AdaBoost
9	84.69	0.1	4	100	AdaBoost
10	87.08	0.00001	6	100	AdaBoost
11	87.56	0.0001	6	200	AdaBoost
12	86.6	0.001	6	100	AdaBoost
13	85.65	0.01	6	400	AdaBoost
14	85.17	0.1	6	500	AdaBoost

Table 11: Accuracy Score for AdaBoost

For each combination of depth and learning rate the table shows the number of base learners that produces the best accuracy score

3.8 Gradient Boosting

The trees used for these Gradient Boosting models have been trained with 2, 4 and 6 branches and the number of trees used goes from 100 to 500, with learning rate from 0.00001 to 0.1. There are two models that produce the same Accuracy Score, one with trees with 2 branches and one with 4. For the same reason as before, let's choose as best model the one that is less computationally expensive, hence the one trained on trees with 2 branches. Moreover, this model presents a high learning rate (0.1) and 200 base learners. The high learning rate means that the steps taken towards the maximum of the function are wide, hence the optimal value was far away from the starting point.



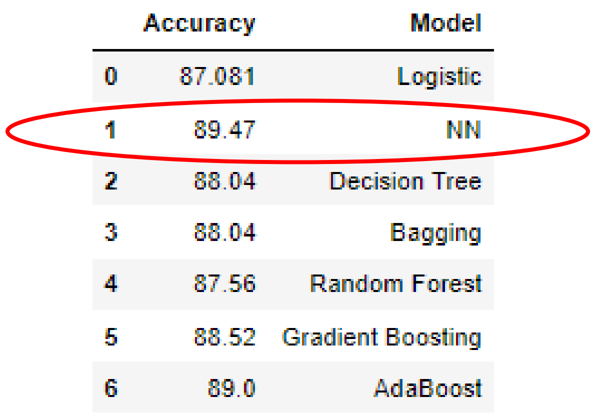
	Accuracy	Learning Rate	Max Depth	Base Learners	Model
0	60.29	1e-05	2	100	Gradient Boosting
1	60.29	0.0001	2	100	Gradient Boosting
2	79.9	0.001	2	200	Gradient Boosting
3	86.6	0.01	2	200	Gradient Boosting
4	88.52	0.1	2	200	Gradient Boosting
5	60.29	0.00001	4	100	Gradient Boosting
6	60.29	0.0001	4	100	Gradient Boosting
7	84.21	0.001	4	500	Gradient Boosting
8	88.52	0.01	4	200	Gradient Boosting
9	87.56	0.1	4	200	Gradient Boosting
10	60.29	0.00001	6	100	Gradient Boosting
11	60.29	0.0001	6	100	Gradient Boosting
12	85.17	0.001	6	500	Gradient Boosting
13	86.6	0.01	6	100	Gradient Boosting
14	86.6	0.1	6	500	Gradient Boosting

Table 12: Accuracy Score for Gradient Boosting

For each combination of depth and learning rate the table shows the number of base learners that produces the best accuracy score

3.9 Discussion

Let's now look at the best models for each of the implemented methods and compare which one is the best method for making prediction on this Titanic dataset.



	Accuracy	Model
0	87.081	Logistic
1	89.47	NN
2	88.04	Decision Tree
3	88.04	Bagging
4	87.56	Random Forest
5	88.52	Gradient Boosting
6	89.0	AdaBoost

Table 13: Accuracy Scores of best models for each method

The best model for making prediction on this Titanic dataset is the Neural Networks (89.47%), followed by the AdaBoost (89%), which is the best among the Ensemble methods. The Logistic regression is the model that performed worse. Once again, Neural Networks proved to be the most suited model to make prediction based on a set of given data, as in Project 2.

Table 8 shows that this model is trained on ReLU, has mini-batch size of 64 and 2 hidden layers. The optimal learning rate is small (0.001), which means that the gradient descent moved quite slowly towards the maximum. The regularization parameter, however, is very small (0.0001), which means that the weights are not heavily penalized, which brings the algorithm to converge slower to the maximum, probably because it is not that far away from the starting value.

It is important to deeply understand that each dataset and choice of parameters lead to a different situation and optimal solution. This conclusion does not mean that Neural Networks *always* work better than Logistic Regression, Decision Trees and Ensemble Methods, even though based on empirical evidence, the researchers affirm that MLPs tend to work better in a lot of situations when the aim of the study is predictions. In fact, there are occasions in which Neural Network are not the best options, for example when the purpose is to describe the data or interpret the results.

4. Conclusion

In this project I tested the prediction ability of different types of models: Logistic Regression, Neural Networks, Decision Trees and Ensemble Methods (Bagging, Random Forests and Boosting). The problem was classification and the aim was to predict whether a passenger on the Titanic survived or not to the sinking based on their age, sex, passenger class and ticket fare. I trained different models for each method, changing the values of parameters and looking for the best hyperparameters through grid searches. The results show that Neural Network trained on ReLU, with mini-batch size of 64 and 2 hidden layers was the best, performing slightly better than all the Ensemble methods.

5. References

[1] Hjorth-Jensen M. *Overview of course material: Data Analysis and Machine Learning. Week 45*. https://compphysics.github.io/MachineLearning/doc/web/course.html. Accessed: 2021-12-08

[2] Sam T. *Entropy: How Decision Trees Make Decisions*. https://towardsdatascience.com/entropy-how-decision-trees-make-decisions-2946b9c18c8. Accessed: 2021-12-04

[3] Lutins E. *Ensemble Methods in Machine Learning: What are They and Why Use Them?* https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ec3f9fef5f. Accessed 2021-12-10

[4] Hastie T, Tibshirani R, Friedman J. *The Elements of Statistical Learning*. New York (NY): Springer New York Inc. 2001.Chapters 15-16.

More discussions and codes can be found at my GitHub repository https://github.com/isarositi/FYS-STK4155/blob/main/Project%203_Rositi.ipynb