A prediction model for survival: Titanic Survival project

Joël Haupt, Miquel Sugrañes, Saurabh Nagarkar & Hajar Laccheb

1. Introduction

This project is based on the Kaggle ML competition which requires the building of a model able to classify if a certain person (instance) survived the sinking of the Titanic or not, given a set of features.
This work will consist of the comparison between several classification models, concretely a Logistic Regression model and different Artificial Neural Network approaches, using the architecture of a Multi-Layer Perceptron. In the first sections of this report, the data used and preprocessing techniques are explained in detail. In later sections, we discuss both approaches to build our Machine Learning classifier, as well as the parameters for each. Finally, we show the obtained results with each model, and choose the best to perform the classification on the test data – the aim of the competition. The final results retrieved from Kaggle after this prediction are shown in the last section of the report.
To finish, we expose our most important conclusions from the work performed, and the computational intelligence research field in a wider view.
Attached to this report we provide a python notebook with the necessary code to perform all experiments.

2. Data

The data retrieved from *Kaggle* consisted into two sets: training set and test set. The training set contains 891 labeled examples represented with 11 different features. On the other hand, the test set contains 418 examples without a class label, which are meant to be predicted by our classification model. After the data retrieval, we stored both data sets into two different *pandas* DataFrames, previously to perform the different necessary preprocessing steps. Just to visually show the structure of our data, we represent in the following table how all features are defined in our training data.

Table 1 - DataFrame representation of first 5 rows of training data

index	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.05	NaN	S

a. Preprocessing steps

One important part of our project has been to preprocess the data. The preprocessing steps are crucial in order to obtain a consistent and reliable data set for our model to perform correctly. Data is one of the most important things to consider, if not the most, when performing any Machine Learning task, so we applied several preprocessing steps in order to polish our training data and prepare it for our classifier.
The first thing we considered and dealt with were missing values. Missing values are features that don't have any value for an instance. In table 1, we can see that the "*Cabin*" feature contains several missing values, represented as *NaN*. Missing values can lead to a bad performance of our model since they are missing information in our data and so our classifier cannot learn from it.
In our case, we had 3 features containing missing values in training data. In the following table we can see very visually which were these features and how many missing values each of them had.

Table 2 - Missing values representation for training data

Feature	N° of missing values
Age	177
Cabin	687
Embarked	2

From table 2, it can be seen that feature *"Cabin"* has a lot of missing values, while the others contain much less missing information. For this reason, we decided to delete the entire "*Cabin"* feature and only those instances with no representation for the rest of columns. To do so, we selected the instance indices that had missing values in either *"Age"* or *"Embarked"*, and we performed a controlled elimination of them.
After this first step, our data dropped from 11 features and 891 instances to 10 features and 712 examples. This means a reduction of around 20% of the data, but we considered that it is more important to have a high representative data (without missing information), even if that means to reduce the number of examples it contains. We also didn't want to add bias by replacing missing values with any technique for our training data, so the information we had was as original as possible.

In our preprocessing, we also considered that some features might not be relevant for the classification task in hands. Concretely, in our opinion *"Ticket"* and *"Name"* features don't contribute much to the classification task, since the information they contain is not connected to the rest of features. Even so, we can extract some interesting strings from *"Name"* feature that can define a new and interesting variable: the social title. We can see from table 1 that the social title is written within the name, for example "Mr.", "Mrs.", "Miss.", etc. This information can be relevant for the classification model since the social status may affect to the survival of an individual, so we extracted these social titles and saved them in a new feature called *"Title"*. Moreover, as some titles could be clustered together, we defined the groups *royalty* and *officer* for those titles related with the royalty, such as *"Sir"*, *"Lady"*, …, and with the military, such as *"Capt"*, *"Major"*, …, respectively.

Our data is made of both categorical and numerical data. Moreover, the numerical data have different ranges depending on the variable. This another issue that our preprocessing had to cover, and to deal with it we performed two more steps, the first one consisting of encoding categorical features into numerical ones, and the second one consisting of arranging the features into similar numerical ranges.
To encode data, we performed a simple label encoder, where each different categorical value is mapped to a numerical one. For example, in the case of the *"Sex"* feature, we decided to set an integer "0" to those instances with the value "*female"* and to set an integer "1" to the instances with value "male". By doing so, we turn the categorical variable into a numerical one, concretely a binary one, containing only 0's and

1's depending on the case. Similarly, we encoded the rest of categorical features: *"Embarked"* and *"Title"*. In these two final cases since the number of different values we had were higher than two the numerical variables obtained after the encoding were multinominal instead of binomial (values of [0, 1, 2] for the *embarked* feature and values of [0, 1, 2, 3, 4, 5] for the *title* case).
Finally, to arrange the feature values we performed a min-max normalization to features *"Age"* and *"Fare"*, since their ranges were [0.42, 80] and [0, 512.33] respectively. The min-max normalization can be performed using the following equation:

$$x_i = \frac{x_i - \min(f)}{\max(f) - \min(f)}$$

Where x_i is the value of the ith instance of the feature f, and the $\min(f)$ and $\max(f)$ are the minimum and maximum value of the instance f respectively.
Moreover, before the training of the models we also performed a standardization step where data is scaled in order for the values to have 0 mean and 1 unit of standard deviation. In other words, data values are centered around the mean within a circle of 1 unit radius. This final step greatly impacted the performance of the training and converged with much less iterations or epochs.

b. Train/Validation data split

Before building and training our models we must took into consideration the fact that we needed some validation data in order to test the performance of the different models. Since we were only able to retrieve train and test data from *Kaggle*, we had to split the training data (containing labeled examples) into train and validation data sets. To do so, we used and already implemented function in *python* language that basically splits randomly the input data and, with a defined size, returns two different sets of data.
In our case, after the data was randomly sorted, we kept the 80% of it to create the test data set and the 20% left for the validation data set.

c. Test data preprocessing

Some previous steps performed in order to preprocess the training data can be also performed to preprocess the test data, but some of them cannot. In the test data we can't delete instances containing missing values, since this would mean that we are not classifying some instances (those deleted instances wouldn't have a class label prediction since they wouldn't exist at the classification time), so a different approach must be used in order to deal with missing values in test data. After performing some analysis, we detected 76 missing values in feature *"Age"* and 1 missing value in feature *"Fare"*.
In order to fill the missing values of the age column which contained 177 *NaN* (Not a Number) values, we considered several options, namely mean or median values or the Expectation-Maximization (EM) algorithm. Looking at the distribution of the "Age" values (Figure 2), they appear to have a similar shape like a Gaussian Normal distribution that's the reason why for our work, we decided to fill missing values in these features with an Expectation-Maximization model.

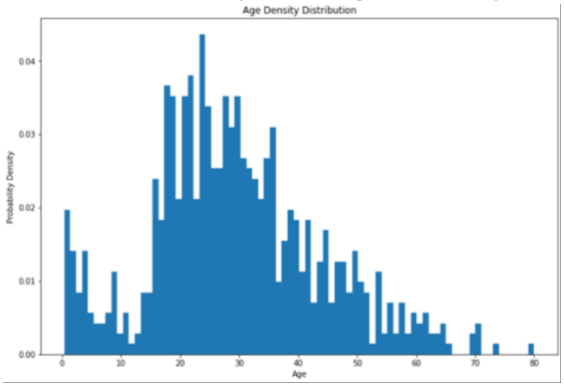

Figure 1 - Distribution of "Age" values

In the context of filling missing values, the EM algorithm estimates the missing values by using the observed data and the estimated parameters from the model. The algorithm starts with an initial guess for the missing values, then it uses the observed data and the current estimates of the parameters to estimate the

probability of the missing data (expectation. The algorithm then uses these probabilities to update the estimates of the missing values (maximization), and repeat the process until the estimates of the missing values converge. It can be used when the data is expected to come from a normal distribution and the values are assigned randomly.
The same approach was used in the *"Fare"* feature to estimate the missing value it contained.
The rest of the preprocessing steps can be considered the same, since they don't affect to the number of instances, with one only exception: when we define the social title and we encode the categorical values into numerical ones, there is a missing value appearing. To solve this, and since the instance that contains the missing value is a grown woman, we encode it as "Mrs."

2. Classifiers

The task of the project was to classify new and unseen examples with a machine learning model. In order to obtain the best results, we decided to build several models, and compare the performance of them with the previously defined validation data, so finally the best model would be used to predict outputs for the test set. In this work, we implemented two main classifiers: a Linear Regressor and a Multi-Layer Perceptron (ANN classifier).

a. Multi-Layer Perceptron (MLP)

A multi-layer perceptron (MLP) is a type of artificial neural network (ANN) that is composed of multiple layers of artificial neurons. The layers are fully connected, meaning that each neuron in one layer is connected to all of the neurons in the next layer. The input to the network is passed through the layers, with each neuron performing a simple mathematical operation on the input it receives from the previous layer, known as activation function. The output of the final layer is the output of the network. MLPs are commonly used for supervised learning tasks such as classification, as in our case, and also regression.

i. Complexity of the network

We have tested two different networks based on their complexity, that is, the number of hidden layers and hidden neurons they have. Both architectures use the *ReLU* activation function for their input and hidden units, and the *Sigmoid* activation function for their output layer. Both approaches also share the optimizer (*Adam*) and the loss function (*binary cross-entropy)*. In the following table all details about the neural networks implemented are shown.

Table 3 - Summary of Neural Networks Architectures

	Architecture 1	Architecture 2
N° of hidden layers	3	6
N° of hidden units	(32, 16, 8)	(256, 128, 64, 32, 16, 8)
Input layer activation function	ReLU	ReLU
Hidden layers activation function	ReLU	ReLU
Output layer activation function	Sigmoid	Sigmoid
Loss function	Binary Cross-Entropy	Binary Cross-Entropy
Optimizer	Adam	Adam

We can see from previous table that for our first approach, we defined a neural network containing 3 hidden layers with 32, 16 and 8 neurons each, while on the other hand, for our second architecture which is the most complex one, we have defined 6 hidden layers with 256, 128, 64, 32, 16 and 8 units each.

The following figures show a visual representation of both architectures, and it can be seen how the second one is more complex than the first one since the number of weights and parameters is much higher because we have more hidden layers and hidden units than in the first architecture.

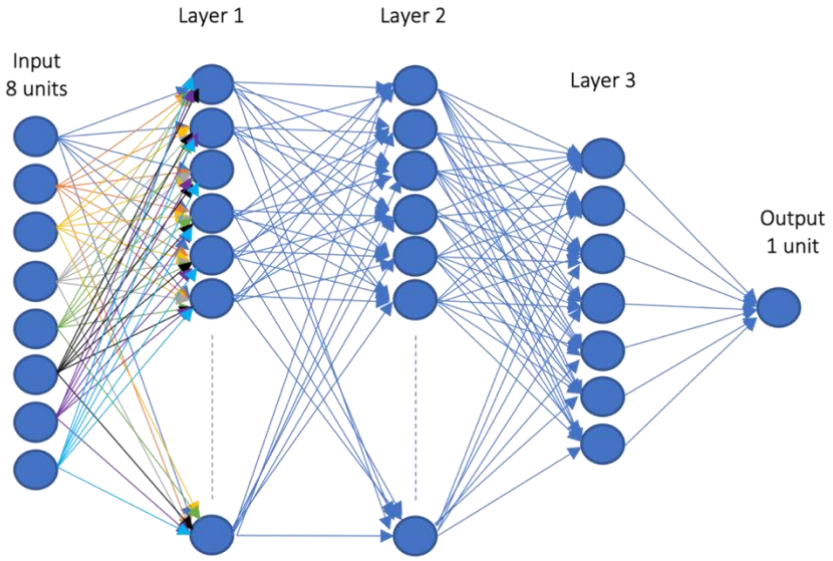

Figure 2 - Architecture for 1st neural network with 3 hidden layers

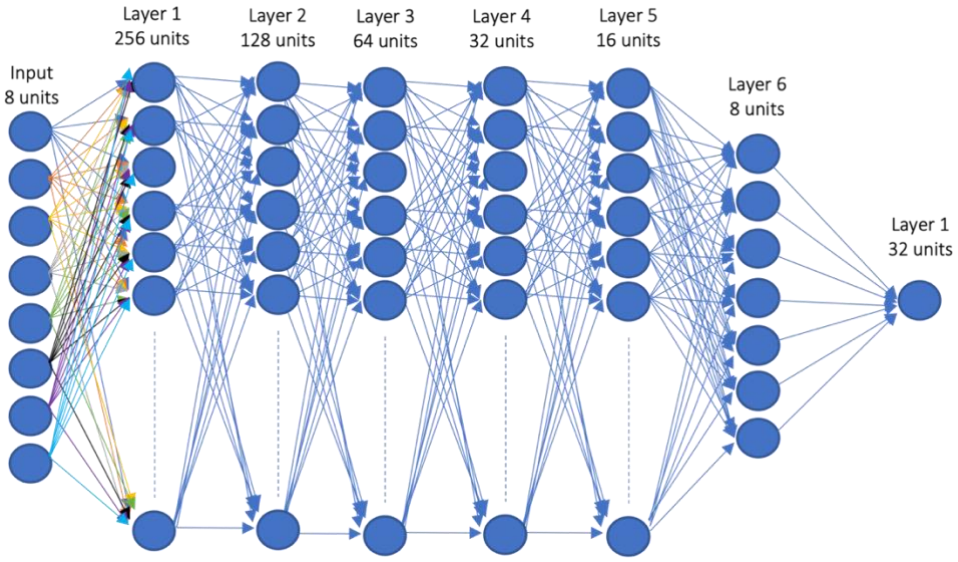

Figure 3 - Architecture for 2nd neural network with 6 hidden layers

ii. Activation functions

The activation functions for our neural network have been defined taking into consideration the type of problem we faced. In our case, we must provide a classification model, so we must choose those activation functions that are better suited to accomplish the correct classification of examples.
For our hidden layers, we made use of the Rectified Linear Unit function (ReLU). This function has been defined as the default activation function when building a multi-layer perceptron model, as it is simpler than other examples and helps both in the speed of training and in overcoming the vanishing gradient problem that other functions suffer from. The ReLU function describes a very simple idea: any input greater than 0 (positive) will be outputted directly, while any input lower than 0 (negative) will be set to 0 for the output.

In any case, probably the most important activation function is probably the one defined in the output layer. This layer is responsible of the final definition of the output values, and so the activation function used in it is crucial for that. The best activation function for the last layer in our MLP that classifies the input into a binary class (Titanic dataset) would likely be the sigmoid activation function. The reason for this is

that the sigmoid function is often used in binary classification problems, as it produces output between 0 and 1, which can be interpreted as the probability of the input belonging to a certain class. In the case of the Titanic dataset, the sigmoid function would be used to output the probability of a passenger surviving or not.
We can also see in later sections that this function is the one used in the logistic regression approach.

iii. Loss function

The best loss function for a binary classification problem such as the Titanic dataset would likely be binary cross-entropy loss. This loss function is commonly used for problems where the output is a probability between 0 and 1 and the goal is to differentiate between two classes. It measures the dissimilarity between the predicted probability distribution and the true distribution (i.e., the label) by computing the logarithm of the predicted probability for the true class.
In the following figure we can see how the log loss (binary cross-entropy loss) behaves when the true label = 1 (in our case, that a certain person survived). We can see that when the probability of the predicted label approaches the maximum possible value (1.0) the loss decreases in a logarithmic fashion.

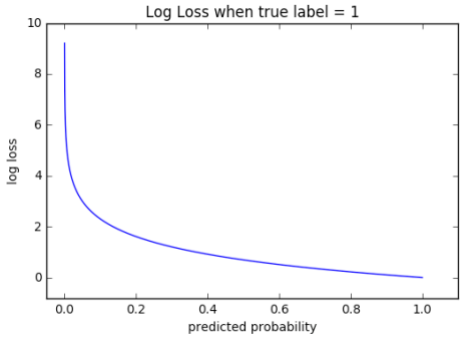

Figure 4 - Log loss plot when true label = 1

iv. Optimization technique

For our MLP approach we used the “Adam” optimization technique to update the model's parameters during training by minimizing the loss function. It is also a commonly used optimizer in deep learning literature.
Adam stands for Adaptive Moment Estimation, and it is a variant of stochastic gradient descent (SGD) that can be more efficient and effective in training deep neural networks than the original methods.
The Adam optimizer computes adaptive learning rates for each parameter, which means that it can automatically adjust the step size during training. It does this by keeping track of the moving averages of the gradients and squared gradients, which are used to estimate the first and second moments of the gradients.

v. Use of regularization

Regularization is a technique used in machine learning to prevent overfitting, which occurs when a model is too complex and performs well on the training data but poorly on new, unseen data.
In our case, we have performed regularization with two common techniques:

- Dropout: randomly sets a fraction of the input units to zero during training, this forces the model to learn more robust features that are useful in conjunction with many different random subsets of the other units.

- L2 regularization: also known as Ridge, adds a penalty term to the cost function that is proportional to the square of the weights, to punish large weights.

In our work the difference of the use of regularization (Dropout or L2) against a model without any regularization term can be seen in the following examples.

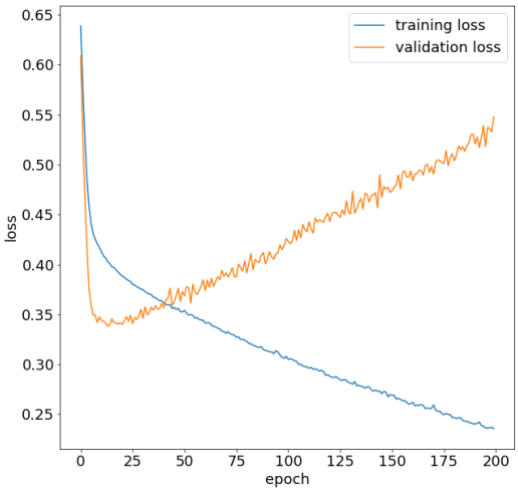

Figure 5 - Training and validation loss for our 3 hidden layer MLP architecture without regularization

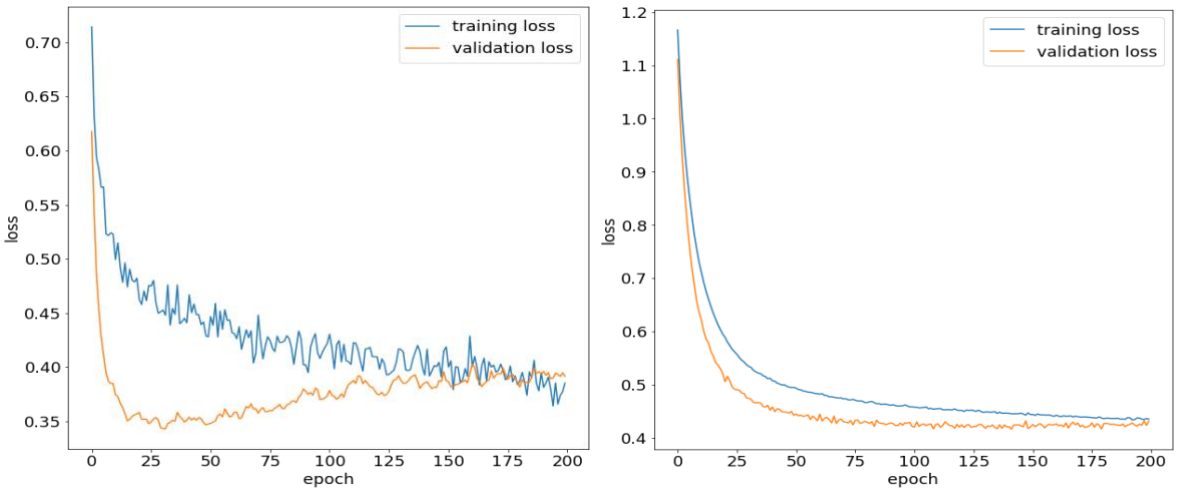

Figure 6 - Train and validation loss over epochs for our 3 hidden layer MLP architecture with Dropout and L2 regularization

From figure 6 we can see that using our MLP architecture with 3 hidden layers and without any kind of regularization the model clearly overfits the data, having high validation error and low train error. On the other hand, we can see in figure 7 that the use of regularization clearly solves this issue, using either Dropout or L2. Even so, we can see that the use of L2 regularization technique (figure 7 – right plot) the performance of the model is better in the sense that the graph obtained is smoother, achieving similar results to those obtained with Dropout normalization, which are lower.

b. Logistic Regressor

Logistic regression is a type of supervised machine learning algorithm used for classification. It is a statistical model that is used to predict a binary outcome (1 / 0) given a set of input features. The model is based on the logistic function (also called the sigmoid function) which is an S-shaped curve that maps any real-valued number to a value between 0 and 1, as can be seen in the following figure.

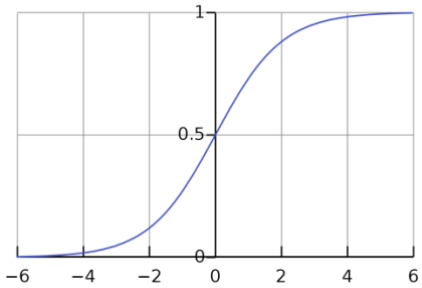

Figure 7 - Sigmoid function

The logistic regression model estimates the probability that the input examples belong to a certain class, and then makes a prediction based on that probability. The model is trained using the labeled training data, and the goal is to find the best set of parameters (also called weights) that maximizes the likelihood of the observed data. To do so, and as explained in the previous sections, we also used a L2 regularization or penalty term so to avoid overfitting and increase the performance of the model. Moreover, by default a value of 1e-4 is used as early stopping term. This means that a descent in the error value is less or equal than this threshold, the training stops. It can also be seen as a measure to stop overfitting and thus to simplify the final model.

In our work, and in order to choose the best number of epochs for the model to learn, we decided to train the regressor with different number of iterations and check its performance against the prediction on the validation data examples.

The performance was obtained with the accuracy measure, which basically computes the number of correctly predicted examples. We could see from this analysis that with a really low number of epochs (i.e.: 20 iterations) the model already converged, and we could obtain the maximum accuracy. Increasing the number of iterations didn't show any increase in the accuracy result, so we decided to keep it in 20 epochs, so the computational cost is also lower.

The following figure is a representation of the confusion matrix obtained, showing the behavior of our model:

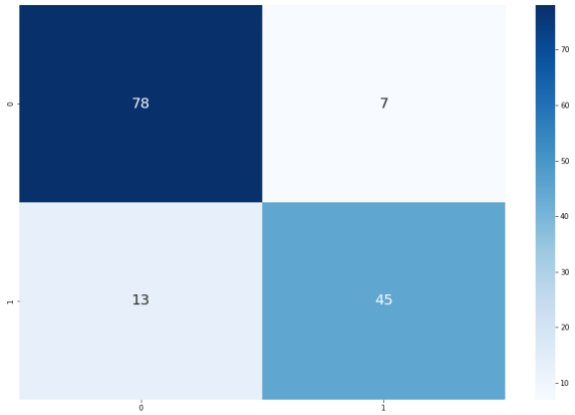

Figure 8 - Confusion Matrix with Logistic Regression results

3. Results obtained and model comparison

In this section we will discuss the different models defined for our task and the results obtained with them when classifying examples from validation data, with the aim of finding the best model for the Titanic classification which will be used to predict class labels for the test data set examples.
Recalling the previous explanations and definitions, we finally have two different approaches to build our models, the first one is using a Logistic Regression technique, and the second one using an artificial neural network, concretely, a Multi-Layer Perceptron. This last approach can be used to build different models depending on the architecture or the regularization term used, and in our case, we defined 4 different models.
The summary of the final models used for the Titanic task in our work can be found in the following table:

Table 4 - Summary of models created

Logistic Regression	**Multi-Layer Perceptron (MLP)**	
	3 hidden layers	
	L2 regularization	Dropout regularization
	6 hidden layers	
	L2 regularization	Dropout regularization

In order to train the models based with the MLP approach, we defined the number of iterations to 50. In figure 8 we can see that setting the number of epochs to 50 is a high enough value for our model to obtain a high (approximate to the maximum) accuracy measure, since the curves flatten even having some fluctuation.

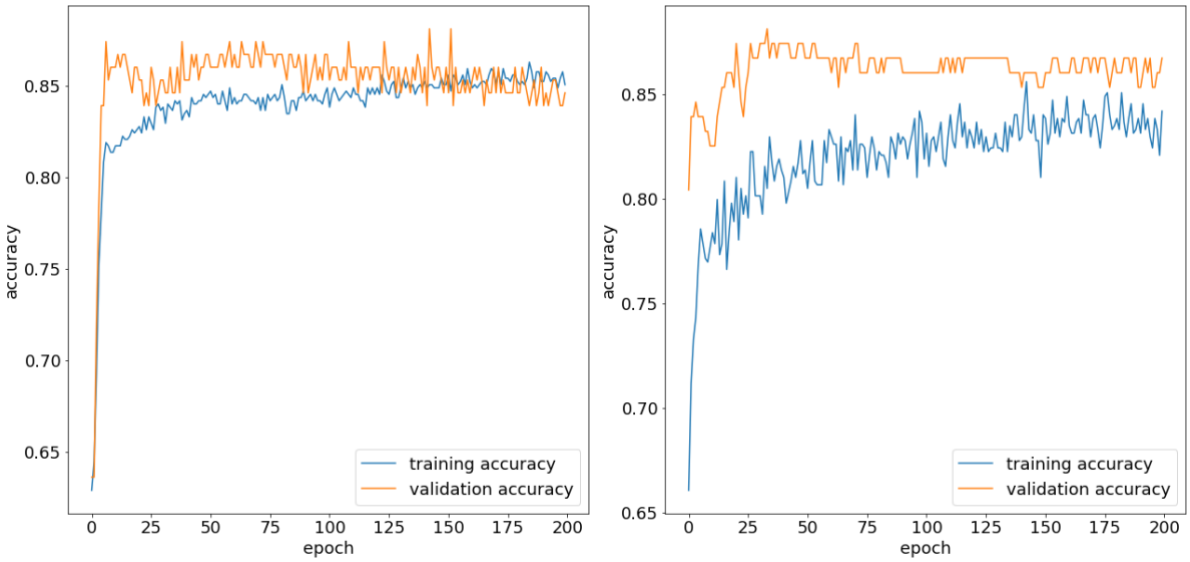

Figure 9 - Train and validation accuracy over epochs for our 3 hidden layer MLP architecture with Dropout and L2 regularization respectively

From previous sections we also defined the number of epochs for our Logistic Regression model to 20, so the next step was to train the models with the training data and to check their performance on the validation data.

The results of the accuracy obtained on validation data with each one of the models is shown in table 5.

Table 5 - Accuracy results on validation data

	Accuracy on validation data
Logistic Regressor	**86.01%**
MLP	
3 hidden layers	
L2 regularization	**86.71%**
Dropout	**86.71%**
6 hidden layers	
L2 regularization	**84.62%**
Dropout	**83.92%**

From previous results (Table 5) we see that even the models behave quite similar in terms of accuracy, the best one is the MLP approach with 3 hidden layer architecture. It performs best independently of the regularization technique used (Dropout or L2)**.** For this reason and since they are the models with the highest accuracy, we decided to use them both to predict labels for the unseen test data.
Moreover, we see that in any case the use of a simpler architecture outperforms the use of a more complex MLP (3 hidden layers vs 6 hidden layers), as well as the logistic regression method that with an accuracy around 86% performs really similar than the MLP approach with 3 hidden layers and is the second-best model considering the tie explained before.
Finally, the chosen models showed a 77.99% performance on test data using Dropout regularization and a 78.47% with L2 regularization when predictions were uploaded to *Kaggle*. We can say that even performance was lower than in train and validation data, which was expected, both of our models performed great on unseen data and thus it was able to generalize, and we can see that the use of L2 actually performed slightly better than the Dropout approach with a difference of less than 0.5% of accuracy. In any case, we can say that we obtained good results given the low amount of training data.

4. Conclusions

The task of classification is a widely known task of Machine Learning problems. To solve a classification problem, we have seen that we can build several different models, depending on the approach and the parameters defined for each. This means that one problem can have several different solutions, and that depending on the approach more than one solution can be valid. Probably, there is no unique optimal solution for our task, but in this work, we wanted to compare the use of Artificial Neural Networks against other classification methods, concretely the Logistic Regression. We have also discussed how the use on Artificial Neural Networks (as MLP) can widely vary depending on the architecture that we give to the model, as well as the different optimization or regularizations techniques. This last one, has proven to be significantly important, even crucial for a model to behave properly and to avoid overfitting the training data and thus, not being able to generalize well. Moreover, the preprocessing of the data is also a relevant task in any computational intelligence work since raw data can contain noisy events (such as missing values) which may lead to bad learning of the model.
Finally, we would like to emphasize that computational intelligence is a fascinating and state-of-the art research field that can contribute to the solution of crucial problems that affect our lives every day, such as climate change, medical diagnosis and treatment, industrial efficiency, etc.

5. Bibliography

Brownlee., J. (2019). *A gentle introduction to the Rectified Linear Unit (ReLU)*. Machine Learning Magestry. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

Cox., D. R. (1958). *The regression analysis of binary sequences*. Journal of the Royal Statistical Society: Series B (Methodological), 20(*2*), 215–232. https://doi.org/10.1111/j.2517-6161.1958.tb00292.x

Romero., E. (2022). Slides-Multilayer Perceptron, *Computational Intelligence class material*. Universitat Politècnica de Catalunya. https://raco.fib.upc.edu/home/assignatura?espai=270700

Romero., E. (2022). Slides-Experimental Issues-Neural Networks, *Computational Intelligence class material*. Universitat Politècnica de Catalunya. https://raco.fib.upc.edu/home/assignatura?espai=270700

Godoy., D. (2018). *Understanding binary cross-entropy / log loss: a visual explanation*. Medium. https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

Haykin., S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR. https://cdn.preterhuman.net/texts/science_and_technology/artificial_intelligence/Neural%20Networks%20-%20A%20Comprehensive%20Foundation%20-%20Simon%20Haykin.pdf

Mukerjee., K. (2021). *Missing value treatment using EM algorithm — A comparative study*. Medium. https://kushalmukherjee.medium.com/missing-value-treatment-using-em-a-comparative-study-e4b0d6c9da61

Maheshkar., S. (2022). *What Is Cross Entropy Loss? A Tutorial With Code*. Weights & Biases. https://wandb.ai/sauravmaheshkar/cross-entropy/reports/What-Is-Cross-Entropy-Loss-A-Tutorial-With-Code--VmlldzoxMDA5NTMx#what-is-cross-entropy-loss?-

Nagpal., A. (2017). *L1 and L2 regularization methods*. Towards Data Science. https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c

Pramoditha., R. (2022). *Creating a Multilayer Perceptron (MLP) Classifier Model to Indentify Handwritten Digits.* Medium. https://towardsdatascience.com/creating-a-multilayer-perceptron-mlp-classifier-model-to-identify-handwritten-digits-9bac1b16fe10

Tan., E. (2022). *How to fill Missing Values with pandas*. Medium. https://towardsdatascience.com/how-to-fill-missing-data-with-pandas-8cb875362a0d