Erasmus Mundus Joint Master Degree in Nuclear Physics

Université Caen Normandie

Machine Learning Project II: Classification Methods and Neural Networks

Project Report presented by

Alan Cíntora de la Cruz & Iván Mozún Mateo & Guillermo Ortega-Ureta

France, January 2023

Abstract

For this Machine Learning project, we move forward to the study of Neural Networks, more precisely, feed-forward neural networks (NN). To do so, we consider two different datasets: the Boston Housing dataset and the Hotel Reservation dataset. The first one was previously analyzed via linear and Ridge regression, whereas here, we explore the Principal Component Analysis as a previous variance analysis before the training of a NN. The second dataset, is studied as a classification problem with 3 different models: logistic regression, k nearest neighbors and a NN. We also do a preliminary analysis of such dataset in order to achieve better efficiency in the models.

Contents

1 Introduction

In our first Machine Learning Project, we put in practice our knowledge on linear regression methods and resampling techniques, applying them firstly to Franke's Function and then to the Boston Housing dataset, in order to evaluate the precision and effectiveness of each possible combination.

The aim of this second project is to delve a step further into Machine Learning methods, this time devoting exclusively to using real datasets. Therefore, the first part of it has consisted in picking up where we left off with the Boston House dataset. Starting with the same assumptions we made in Project I in order to simplify our dataset, this time we apply a Principal Component Analysis method in order to more properly justify our choice of the most important variables for our method while keeping the maximum possible amount of accuracy.

After that, a **Multi Layer Perceptron (MLP)** was implemented as our main tool to work with the pre-analyzed data. This is a type of feed forward, fully connected Neural Network which consists on an input and an output layer, and a certain number of hidden layers in between. This hidden layers use non linear activation functions in order to fulfill the *Universal Approximation Theorem*, that enables them to approximate any continuous, multidimensional function, which is our objective with this dataset. In order to reduce overfitting, an extra term is usually added to the cost function, analogously to what is done in Lasso regression. This term is proportional to the size of the weights, thus enables us to put a limit on how much they can grow, via the L2 regularization parameter, which we call α

With the objective of finding the optimal configuration for our MLP, several combinations of parameters were tested out, subsequently calculating the typical statistical estimates that give us an idea of the quality of our model. The three most typical activation functions were added as options, namely a logistic curve, a Rectified Linear Unit and a hyperbolic tangent. Various options for the architecture of the network were also included, with either two or three hidden layers and with different numbers of nodes for each of them. Furthermore, an array of diverse values for the L2 regularization parameter we also tried out.

Working with the Boston housing dataset allows us to solve an interesting regression problem. However, it was also compelling for us to get involved with a classification problem, as a way to put in practice the different methods that we have learned in a context with discrete variables. For that, we chose a Hotel Reservations dataset, which includes 17 different parameters related to reservations in hotels (number of adults and children, arrival date, meal plan, etc.) and in which the objective is to know if a certain reservation will or will not be canceled.

The first step in this second part of the project is also to do a pre-analysis of the data. Some of the parameters, such as the room type, were defined by words, which cannot be processed by our algorithms. Therefore, we begun by translating this words to numerical values. Then, with the objective of reducing the amount of options that will have to be taken into account by our algorithms, thus lowering computational time, we grouped some of the classes that had lower statistics (for example, the classes for 2, 3, 4 and 5 children were grouped in a single '2 or more children' class, and the different values for 'average price' where grouped in price ranges). Finally, the correlation between the different parameters was also studied, as it was done with the Boston Housing dataset.

After this pre-analysis, three different machine learning models were implemented: Logistic Regression, k Nearest Neighbors and, of course, a MLP Neural Network. **Logistic Regression** is the simplest, most straightforward model that we can apply to a classification problem. It consists on the usage of the logistic function (the same one we can use as activation in an MLP!) as the probability distribution of our data points being part of each of two possible classes. We aim to fit our data to this distribution in a certain number of iterations given a certain tolerance value. On the other hand, the **k Nearest Neighbors** algorithm falls in the more complex category of supervised learning methods, which rely on the training data being labeled in order to then find the adequate label for the test data. This algorithm in particular consists in assigning a class to a certain data point based on the class labels of its nearest train data points, assuming that data belonging to the same class is close together.

Of course, each of the three used methods has a number of parameters than can be optimized with the objective of obtaining the maximum possible likelihood between our predictions and reality. This likelihood can be inferred from the score function statistical estimator, which was evaluated for every parameter combination until the best one was found. For logistic regression, the maximum number of iterations given to reach the tolerance was tuned, and in the case of k nearest neighbors, the tested parameters were the amount of neighbors considered, as well as the weight given to each of them (with the options of given them all equal influence or, in contrast, giving more influence to a neighbor the closer it is). Moreover, we know the rescaling of the values is of great importance for this two methods. In consequence, three rescaling options were also tried out, them being no rescaling at all, a renormalization of the values to the [0,1] interval, and scaling them with a gaussian distribution with $\mu = 0$ and $\sigma = 0$.

Finally, the same score function estimator was calculated for the final, optimized versions of each method, with the purpose of comparing them and finding which one is best suited to our classification problem.

2 Description of the code

The code that we developed for this project can be found in our GitHub repository, contained in the two Jupyter Notebooks named ”Machine Learning Project 2”, which correspond respectively to the two parts of the project as described in the introduction section.

2.1 Python Libraries

This Machine Learning project was developed in the Python programming language, one of the most widely used languages in this field, due to its simplicity, conciseness and specially to its great flexibility. Python has a wide array of libraries full of functionalities to make our coding work easier and more straightforward, as well as keeping our code shorter and much more readable. The main libraries that we have included in this project are the following:

- `numpy`: Very complete library from which we will mainly use its various mathematical functions, support for vector and matrix creation and linear algebra routines.

- **`matplotlib` and `seaborn`**: This two libraries include every tool we need for making plots, graphs, heatmaps and tables. While `matplotlib` is more of an all purpose plotting library, `seaborn` focuses in statistical data visualization, making more specific functions available.

- `pandas`: Data analysis oriented library from which we are mainly interested in its data manipulation features, ideal for reading datasets and they shaping them to our convenience.

- `sklearn`: SciKit-learn is a machine learning oriented library from which we use some of its predictive data analysis tools, such as tools for normalization, train-test splitting, resampling methods and calculation of statistical parameters. This time we also use its functionalities more directly related to the new methods that we are applying (logistic regression, kNN and MLP), as well as some extra metrics that will be useful to extract conclusions from our work.

Even though at the present time a variety of good machine learning libraries exist, such as Pytorch or TensorFlow, we chose to stick with SciKit-learn for this work, considering not only that is one of the most widly used ones, but also that we already had some experience with it from Project I, and we deemed it very complete feauture-wise, as well as intuitive to use.

2.2 Boston Housing Dataset with NN

After the library definition, the code starts by extracting the Boston Housing dataset directly from the very practical SciKit-learn database. We will be using it as a `pandas` DataFrame, defining the MEDV parameter as our target. From there, we can build the correlation matrix of our dataset, plotting it in a heatmap and hence getting a very visual representation of the correlations between the different parameters.

As explained in the introduction, we performed a pre-analysis of this data analogous to the one from Project I, that follows the same two assumptions:

- There are three features that have a correlation with our target value of 33% or less, namely "CHAS", "DIS" and "B". Therefore, we can discard them from our analysis, as their usefulness will not be as big as the computational complexity they add to the analysis.
- The features "AGE" and "RAD" have the same correlation value with our target. Thus, as they give us the same information regarding our analysis, we discard one of them ("RAD") to again reduce the complexity of our problem.

However, this time we performed a Principal Component Analysis (PCA), which allows us to asses wether our assumptions were reasonable, and the exact impact of eliminating those parameters on the accuracy of our global analysis. For that purpose we used the pipeline functionalities of SciKit-learn, rescaling our data with the StandardScaler option and then applying the PCA, which turns the 13 parameters of our model into 13 principal components, calculates the importance of each of them to our model and orders them following that criteria. We end by plotting the obtained results in the following graph, that shows the cumulative explained variance ratio, which tells us how much information we lose by just keeping a certain number of parameters. As it can be seen, by reducing our parameters from 13 to 9 we still can explain 95% of our model, which we found an acceptable trade-off for the computational time gain.

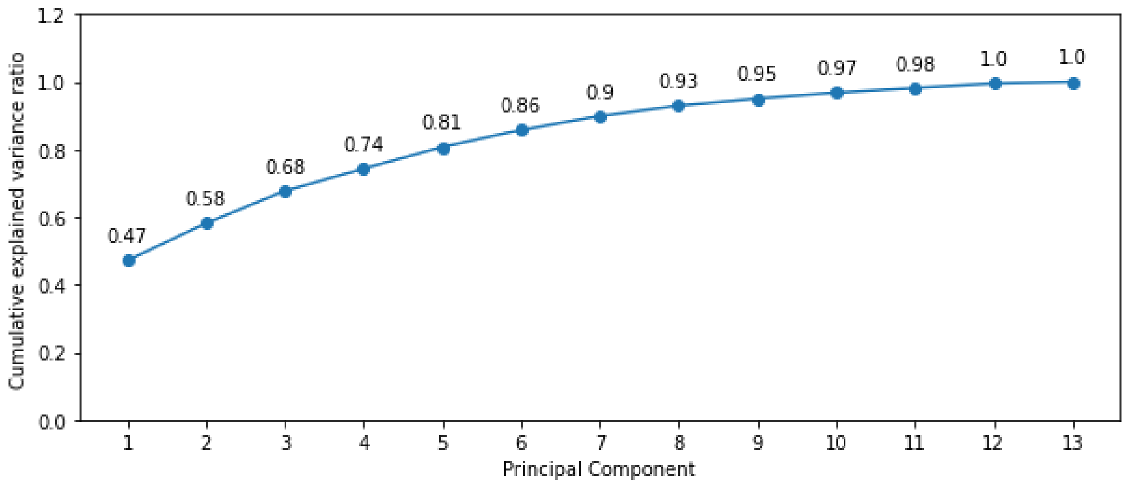

Figure 1: Resulting plot from the Principal Component Analysis

Following this results, the number of parameters was consequently reduces, and the resulting data collection is showed in the table that can be found in the Notebook.

Subsequently, two plots were generated, which illustrate the probability density functions of the target parameter, MEDV, before and after a Yeo-Johnson power transformation, that rescales the data and at the same time reduces asymmetry and brings that distribution closer to a gaussian. This is done since we have used an analysis of variance technique as a part of our PCA which required this type of scaling for our feature parameters, so we applied the same treatment to our target. Therefore, this is the final distribution we will give to our Neural Network.

At last, we arrive to the implementation of the Neural Network. We start by splitting our data, devoting 75% of it to training and 25% to testing, staying at typical values when working with Neural Networks. Then, we set up our MLP with an adaptive learning rate, which allows it to decrease its initial rate if at any point the training loss doesn't go down between iterations. The maximum number of iterations is also set to 5000. After that, the different parameter options explained in the introduction are defined, and finally, the grid search method is implemented, which uses the Cross-Validation (CV) technique to find out which is the best combination of parameter options basing it on the calculated R^2 scores for each of them. We set it to use 3 CV folds, which is a good compromise between high accuracy and low computational time in this case. The optimal parameter set is displayed at the end.

Finally, an additional run with the MLP is performed setting it up with the just acquired ideal parameters, and a final R^2 score is calculated for both the training and the test data.

2.3 Hotel Reservations Dataset

This second part of the code starts again by importing the Hotel Reservations dataset that we will be working with. This time, the information is extracted from a csv file, obtained for the popular Kaggle database.

Then we go on with the pre-analysis of the data, as it was explained in the introduction. We start by renaming every parameter, which improves readability, followed by plotting all of our data before making any modification. This plots, as well as the lists that are displayed after them, were used preliminarily to decide which classes were more suited for adjustments. Then we proceeded with the transformation of *string* into *int* values, and with the class grouping. For the latter, we studied the possibility of directly using One-Hot Encoding, which is implemented as a built-in method in SciKit-learn. However, we realized this approach could increase the number of features, thus hindering our final result. Instead, we decided to manually code this grouping by ourselves using regular python tools, following a philosophy that more closely resembles that of Label Encoding. After each parameter's classes are modified, a new plot of them is presented so the comparison can be seen.

As a final step to the pre-analysis, a correlation matrix was obtained and plotted, but in this case, no whole parameters were dropped. This consideration was made due to this being a classification problem, in which we considered narrowing down classes was much more profitable and efficient than doing so with parameters.

The next part of the notebook is devoted to our three machine learning models. In order to prepare the data from them, we separated both the ID and the 'Booking Status' parameters, the latter being our target in this dataset. Following that, we split our dataset the same way we did in the first part of the project, this time devoting 70% of the data to training and 30% to testing. Then we went on with the analysis.

The first model that is found is the Logistic Regression, then the k Nearest Neighbors and finally, the MLP. The followed procedure is parallel for all of them, and at the same time equivalent to the one we applied for the Boston Housing dataset. The model is set, via a pipeline for the first two instances and directing setting up the MLP in its case, using the same learning rate and maximum iterations we had used before. Then, the different parameter options detailed in the introduction are given for each case, followed by a grid search, this time using 5 CV folds for the three models and basing the choice of optimal parameters on the accuracy score, more fit to a classification problem. Finally, the best parameter combination, as well as its associated accuracy score was displayed.

To wrap up the code, a variety of statistical estimators, as well as confusion matrices are presented for each of the models, with allow us to more profoundly assess the precision of each of our models. The confusion matrices, which can be observed on the results section, are particularly interesting, as their diagonal values give us the amount of true positives (top left) and true negatives (bottom right) obtained with every model, that is to say, the number of correct predictions. On the contrary, the non-diagonal values give us the false positives (bottom left) and false negatives (top right), the incorrect predictions of our model.

3 Discussion of the results and conclusions

In this whole project we have studied the two main types of problems that can be addressed with the usage of machine learning: a regression one and a classification one. We had the objective to figure out not only the best parameters for the different models, but also which of them is the better suited in each of our problems. With the obtained statistical estimates we can extract the desired conclusions.

Starting with the Boston House regression problem, we found the following to be the most optimal choices for each of the parameters:

- **Activation function**: ReLU
- **Regularisation parameter**: 1.0
- **Hidden layer architecture**: 3 hidden layers, with a [90,30,10] node structure.

This parameters allow our model to finally obtained an R^2 score of **93.2%** for the training data, and **86.1%** for the testing data. This is a really important improvement when we compare it with the results obtained in Project I when using simpler lineal regression models, with which the maximum R^2 score was of **77.7%**, corresponding to the Lasso regression case. This was to be expected, as an appropriately tuned neural network is a much powerful tool than a more straightforward linear regression.

Continuing with the Hotel Reservation classification problem, the optimal parameters are presented in the following for each of the three used models:

LOGISTIC REGRESSION

- **Scaling**: None
- **Maximum iterations**: 500

K NEAREST NEIGHBORS

- **Scaler**: Gaussian distribution
- **Number of nearest neighbors**: 14
- **Weight distribution**: Based on distance

MLP NEURAL NETWORK

- **Activation function**: Hiperbolic Tangent
- **Regularisation parameter**: 0.1
- **Hidden layer architecture**: 3 hidden layers, with a [30,20,10] node structure

The accuracy scores obtained with the best configuration are of **77.9%** for the logistic regression, **82.9%** for the MLP and **83.6%** for the k Nearest Neighbors, making the latter the most accurate model for this dataset. Coincidentally, if we take a glance at the computational times the models have taken, we find that the k Nearest Neighbors is also the fastest one. To complement this information we can analyze the confusion matrices in the following figure:

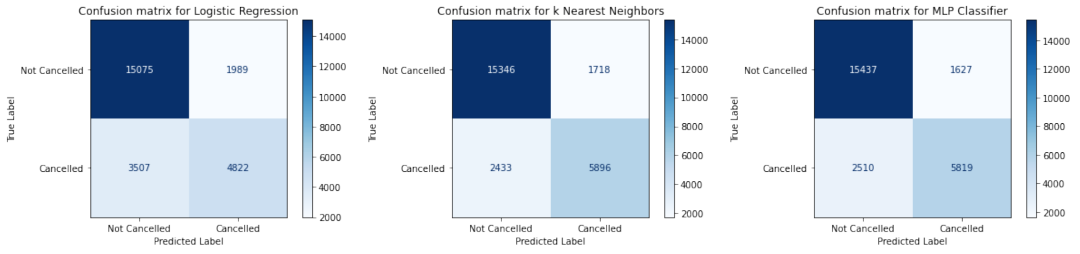

Figure 2: Angular distribution of the final energies for the upwards scattered proton (left) and for the downwards scattered proton (right)

They tell us that, even though the k Nearest Neighbors is overall a better model, it is not by a wide margin, and it and the MLP perform slightly different for false positives and negatives. Therefore, both models might be suitable depending on the specific needs of the interested part.

If we examine this results globally, we can remark that for each of the studied datasets the most accurate model varies, and even when using the same model (in this case, the MLP neural network), we find different optimal parameters in each case. From this, we can draw a global conclusion: when applying a Machine Learning approach to a certain problem, it is of sheer importance to carefully analyze the specific kind of problem that we are facing, as well as the particular dataset we are working with. Then, with a good enough knowledge about the different machine learning methods, one can choose the most appropriate one for the situation, regarding both high accuracy and acceptable computational time.