Machine Learning - 2023 - Mini Project I

Aicha Moussaid
Aicha.moussaid@abo.fi
EDISS Master's Programme - Åbo Akademi University, Turku, Finland

Introduction

A term deposit, also known as a fixed deposit or lump sum deposit, is a type of investment made with a bank where a fixed amount is deposited for a predetermined amount of time at an agreed-upon rate of interest. Instead of keeping a savings account open, customers often establish term or fixed deposits to ensure that their money will increase in value over time and that they can save substantially.

On the other hand, offering term deposits to the clientele is profitable for banks since it allows them to add more money to their cycle. Instead, the bank provides consumers with interest rates that are higher than those offered on savings accounts. In fact, banks engage in promotional efforts, such as phone, email, and advertising campaigns, to entice clients to open term deposits with them.

In this project, we will be applying the prerequisites of machine learning, specifically Artificial Neural Networks, on this banking situation, where we will train a model to predict whether a client would respond positively or negatively to the campaign conducted by a bank.

Data Preprocessing will be applied on the Bank Marketing Dataset used in this project to adjust it for the next step, we then feed it to the neural network model built, train the model, and generate an evaluation of its performance, and analyze its architecture and performance.

Data Processing

The used data is provided by UCI and referred to as the Bank marketing Dataset. It contains 19 input variables that concern the bank client data (age, job, marital, education, default, housing, loan), the last contact of the current campaign (contact, month, day_of_week, duration), the social and economic context attributes (emp.var.rate, cons.price.idx, cons.conf.idx, euribor3m, nr.employed), and more (campaign, pdays, previous, poutcome). It also contains the 20th attribute which is our target output y, which answers to "Has the client subscribed a term deposit", and accepts binary values of "yes" or "no".

In the data processing step, we first performed a global visualization which shows us the distribution of data in different categories, and a heatmap that illustrates the correlations in the data, as shown in Figure1 and Figure 2.

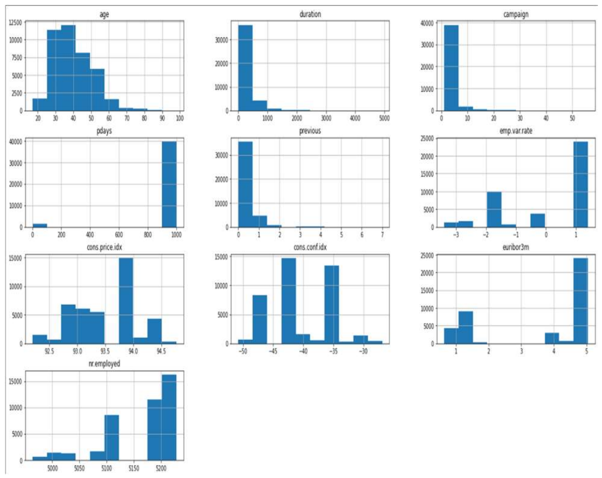

Figure 1. Data Distribution of different dataset attributes

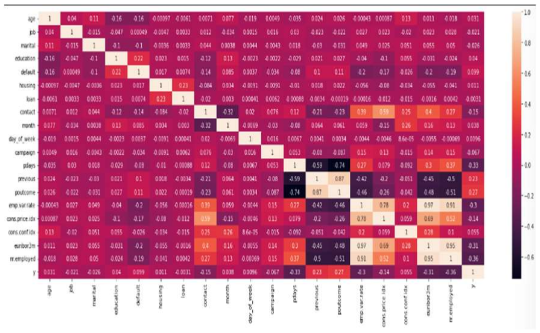

Figure 2. Data Correlation Heatmap

Next, we handled outliers, which are datapoints that deviate a lot from the standard dataset. Having outliers in our dataset when training and building a model effects the ultimate accuracy. Therefore, we have to find and remove such outliers. This was only performed on our numerical features and not the categorical ones.

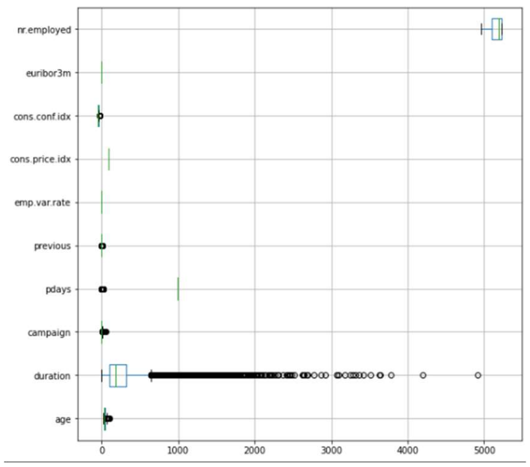

Figure 3. Outliers Visualization

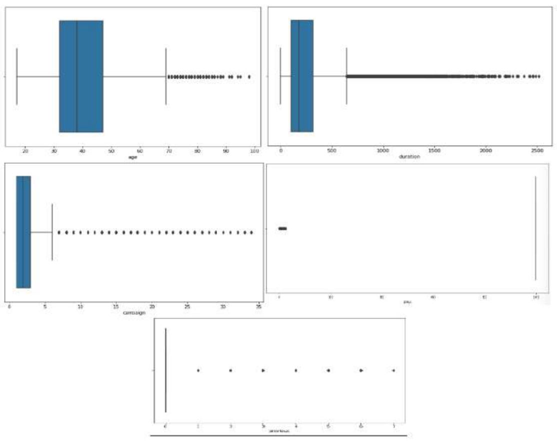

Figure 4. Outliers Visualization After Cutoff

Closer inspection was paid to each attribute since we are trying to avoid eliminating datapoints that could be useful to our mode, like the case for *pdays* attribute, and by identifying appropriate cutoff ranges, we transformed the boxplots shown in Figure 3, to the final results in Figure 4.

The data processing step also included checking for null values, and the balance of the values included, which showed us the presence of the value "*unknown*" across multiple categorical features, but because it is in itself indicative, we opted for keeping it and not considering it as a null value. We also dropped the attribute *duration* since it was stated how it highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call *y* is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

After this, we performed feature encoding of categorical data to a numerical format using LabelEncoder. We also scaled the features since the categorical data encoding output different data ranges, so we standardized them using StandardScaler.

As for the feature engineering section, we referred to a new correlation matrix, then checked for any highly correlated features (>0.95) to remove them, but none were found. We also applied Principal Component Analysis (PCA) to reduce the dimensionality of the dataset by transforming a large set of variables into a smaller one that still contains most of the information in the large set. We also split the data to 80% as a training set and 20% as a testing set.

With this, our data was processed successfully and was ready to be fed to our neural network to start the training.

Modelling

In this section, we built a two-layer neural network which consisted of an input layer with 16 units, and two different Dense layers. We opted for ReLu as an activation function, and sigmoid for the output activation function since this is a binary classification problem. We also used binary_crossentropy for our loss function, root mean square propagation as an optimized and our main metric to be the accuracy. The architecture of the model is shown in figure 5.

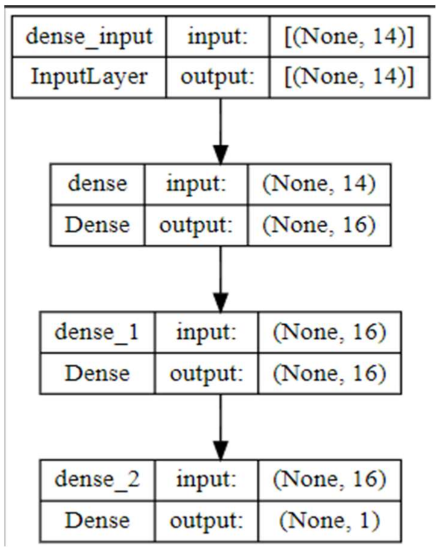
dense_input | input: | [(None, 14)]
InputLayer | output: | [(None, 14)]
dense | input: | (None, 14)
Dense | output: | (None, 16)
dense_1 | input: | (None, 16)
Dense | output: | (None, 16)
dense_2 | input: | (None, 16)
Dense | output: | (None, 1)

Figure 5. Neural Network Architecture

The model was trained using different implementation, where we started by a normal straightforward approach, then we tried using Dropout, Regularization, and early stopping. The performance varied by a small margin, but we decided on settling for the regularization approach, training the model for 5 epochs with a batch size of 100. This shows that even if the margin of difference in performance is very low, it still matters that we explored different implementation and tried to observe the behavior of the loss function and the accuracy metric.

Our model performed with 90% accuracy on the training set and 89% on the testing set, but through the loss and accuracy graphs in Figure 6 and Figure 7, we could say that we avoided overfitting successfully.

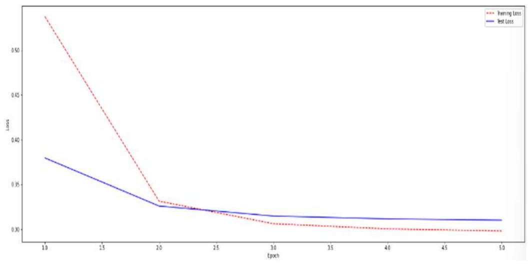

Figure 6. Loss Function Graph

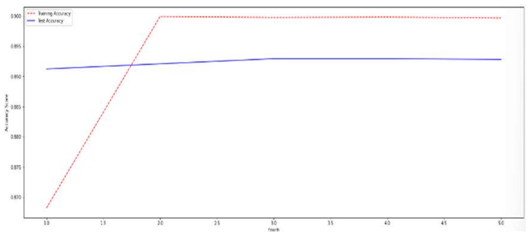

Figure 7. Accuracy Graph

The generated confusion matrix (Figure 8) shows that there are 166 true positives, 797 false positives, 85 false negatives and 7182 true negatives, which is still a good prediction performance for an unbalanced dataset with "no" target values that are 8 times the "yes" class values.

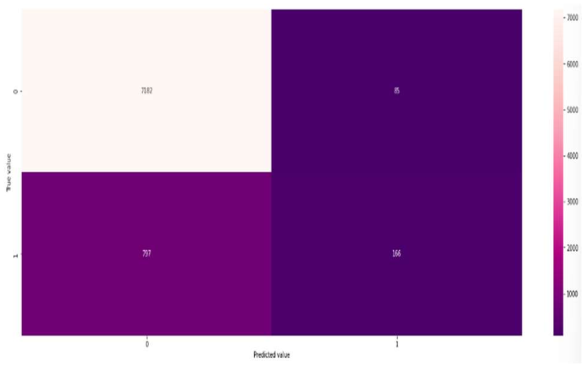

Figure 8. Confusion Matrix of the Neural Network

Conclusion

This project brought up to the table the interesting side of training neural network models and its complex steps. It made it clear that, first of all, managing the data firsthand and processing it before feeding it to the models can make or break the performance. Second of all, the tweaking of the models and their hyperparameters has to be closely monitored to watch its effect on the accuracy of our predictions. Lastly, choosing the correct architecture and techniques to handle overfitting depends solely on the type of dataset you have, how large and balanced it is, the type of problem you are handling, and analyzing the strengths of each available model, which can help us decide the optimal model that can, both, save us time and allow us the best prediction.

Although the unbalance of the dataset was a bottleneck, it leaves room for the scope of improvement and experimenting where we could apply statistical techniques like the Synthetic Minority Oversampling Technique (SMOTE) to increase the number of cases in our dataset in a balanced way.