MINI PROJECT 1 -- REPORT

Author: Junel R.S. Solis

INTRODUCTION

For this mini project, data was provided regarding a direct marketing campaign performed by a banking institution. The data is to be prepared for and processed by a deep learning algorithm in order to predict whether a client would subscribe to the product in question -- a bank term deposit.

DATA PROCESSING

A CSV file containing 41,188 rows of data was read into memory. Processing was required for the categorical and numerical features to be used.

One-hot encoding for features

A number of features were non-numerical and needed to be transformed in order to be processed by the neural network. The columns *education*, *default*, *contact* and *poutcome* were transformed into one-hot columns. The rest of the categorical columns *job*, *marital*, *housing*, *loan*, *month*, and *day_of_week* were dropped from the clean data set in order to reduce overfitting with too many features. The decision of which categorical columns to drop were also based on iterative manual training to see which columns provide higher evaluation accuracy.

Numerical features

The numerical features were normalized using the **MinMaxScaler** provided by the **sklearn.preprocessing** library, with one exception being the *pdays* column.

pdays column

The *pdays* column was transformed using the **StandardScaler** class provided in **sklearn.preprocessing**. The rationale behind this is that apart from the feature being used to count the number of days passed after the client was last contacted from a previous campaign, the number **999** was also used to denote **no previous contact**. This would have presented a problem if transformed using the **MinMaxScaler**. It was thought that using the **StandardScaler** would make the values less affected by outliers, and the training data seems to support this in that higher test accuracies can be achieved when the StandardScaler is used on the *pdays* column compared to the MinMaxScaler.

It was also attempted to change the **999** values into numpy **NaN** but this type of transform would render the loss functions numberless during training.

Encoding the labels

The data labels are stored in the final column called "**y**". The **OrdinalEncoder** class from **sklearn.preprocessing** was used to encode the **yes** / **no** strings to **0** / **1** integers

Histograms

As an aid for previewing data, histograms for the cleaned numeric data were created.

Splitting the training and test sets

The data set was split into training and test sets using the **train_test_split** function provided by **sklearn** with the test data set being 33% of the total data.

MODELLING

A TensorFlow sequential neural network was created via the Keras API. This network had an input layer with 6 neurons, relu activation and an input dimension set to equal the number of columns in the training data.

A second dense layer with 32 neurons and relu activation was configured.

The output layer consists of 2 neurons with softmax activation.

The model was set to use the adam optimizer and its loss function was set to use Sparse Categorical Cross Entropy.

CONCLUSION

When evaluated on a test set, the model achieves an accuracy of **90.1%**.

The model seems to consistently achieve higher accuracy on validation compared to the training; this may be explained by the fact that training accuracy is measured while the epoch is still in progress, while validation accuracy is measured after the epoch is completed.

On the graph of losses for the training and validation sets, there does not appear to be overfitting of the model. The validation curve is lower for the validation compared to the training set. I do not know that this is an issue, but if it is, it could be resolved by adding more data.

Overall, I am happy with the results of this model, considering that this is my first project involving both sklearn, tensorflow and keras. The major bottleneck for me was the learning curve involving machine and deep learning as there are numerous frameworks and libraries that I had to grasp, as well as learning the basic concepts in machine learning and neural networks. I also initially experienced difficulty in making decisions as to how to preprocess the data and this was mainly due to my relative unfamiliarity with Python libraries such as **numpy** and **pandas**.

I was able to overcome these hurdles by referencing various books on machine learning and neural networks as well as searching for answers on the internet. It was also necessary to perform a lot of trial-and-error within the Jypter notebook.