ECM2423: Credit Card Defaults Deep Learning Report

James Thomas
Department of Computer Science
University of Exeter
Exeter, UK
jt799@exeter.ac.uk

I. Abstract

In brief, by using a Sequential Model for building an Artificial Neural Network, with hyper-parameters tuned and appropriate hidden layers added, a credit card holder defaulting on a payment can be predicted to a 78.7% accuracy. This report gives detail to the design and implementation process involved with building such a network, as well as a discussion on recent related literature on Deep Learning.

II. Introduction

This report aims to discuss the process of designing and implementing a Deep Learning technique on the provided dataset for Credit Card Default data.

A Credit Card user defaults on their credit card when they have not made the minimum payment for 180 days, as a result of this the card holder's credit score will plummet [1]; meaning they are no longer as trusted by lenders, which makes applications for loans or mortgages far less likely to be accepted. Depending on the policy of the card, the account may also be closed and the user's wages may be garnished if this results in a lawsuit being filed.

The model I will design and implement aims to predict, based on previous payments on the credit card, as well as context on the card holder, if the user will default on their payment. Using data to reliably predict this is helpful as it would allow lenders to foresee any problems and provide pre-warning to card holders which may help them avoid this happening; thus protecting their credit score.

The dataset has similarities to the "Credit Card Fraud Detection" dataset, which can be found on Kaggle [2]. The Keras documentation shows an example of how this imbalanced dataset can be used for building a Deep Learning model [3]. Author, 'fchollet', designed and implemented their own deep learning model for the dataset, to do so they first vectorised the data and then normalised it such that the data is more useful. It can be seen that for their model they used a Sequential architecture with Dense layers using the 'ReLU' activation function, some dropout layers, and then a final layer using the 'Sigmoid' activation function. I will keep this structure in mind throughout my design process as a strong network was yielded with this architecture.

In the paper titled 'A Classification Approach of Neural Networks for Credit Card Default Detection' [4], a direct example can be seen where an ANN has been used to detect credit card defaulting. In findings from this report, a Neural Network was implemented to achieve a best accuracy of 79.97% for prediction. The main takeaway from this report is that an Artificial Neural Network can be an appropriate network to build to detect credit card defaulting. The report also gives me an approximate target accuracy metric.

This report is structured by first outlining my 'Proposed Method' for developing the Deep Learning technique, this section will cover the pre-processing to be applied to the data as well as a plan for the architecture and how I propose to use Validation to develop the model. 'Experimental Results' will cover my results from implementing the proposed model and show the process of optimising the model. The final 'Summary' section will cover the results of the report and make comment on the appropriateness of my model.

III. Proposed Method

A. Pre-Processing

The most simple way to pre-process the data is to simply load it into the program using a Pandas dataframe and then use the Sklearn 'train_test_split' function to split the data into test/train sets with an 80:20 ratio.

Whilst the above may be effective, issues may arise with an imbalance of samples for each of the outcomes of the data. Through inspection of the data it can be seen that a '1' output occurs 6,636 times in the 30,000 entries; '0' occupies 23,364 of the results, this equates to '1' representing just 22.12% of outputs. To combat this I will test using both over and under-sampling methods.

Undersampling will be tested using Random Under Sampling (RUS) which will remove random data points with a '0' output, thus reducing the imbalance of the overall dataset. This will lead to a reduction in the amount of training data available and may result in loss of accuracy in the model because some valuable data may be removed. Despite these drawbacks, it is valuable to test a range of solutions to re-balancing the data.

Oversampling can either be implemented and tested with Random Oversampling (ROS) [5] or with Synthetic Minority Oversampling (SMOTE) [5]. An ROS approach will duplicate minority records to re-balance the output instances. SMOTE is more complex, it works by creating synthetic data for the

minority class to re-balance the dataset. Both of these over-sampling techniques will be explored during implementation.

Inspection of the dataset shows that there are no null or 'NaN' values, this means there needs to be no handling of null values in the data.

Another problem with the raw data is the measure of spread between the columns, to resolve this the data will be Normalised in the pre-processing stage. Z-Score Normalisation [6] is a process through which data is transformed such that the Mean (μ) and Standard Deviation (σ) satisfies $\mu = 0$ and $\sigma = 1$. By doing this, the layers of the Neural Network will not discount certain data points that fall outside of a range and all data columns becomes valuable to the network. During testing of the model design I will begin by testing without the data normalisation and then test with it, this should validate the claim that normalisation is a valuable pre-processing technique for the data. The 'Imblearn' library has a 'StandardScaler' class built in which I will make use of to handle the Z-score normalisation.

Finally, One-Hot Encoding (OHE) will be implemented for the 'SEX' column of the dataset. Currently this field is occupied by values of '1' and '2' to represent each possible sex in a numerical manner. This isn't useful for the network and will likely be disregarded; instead I will create two new columns, one to represent sex '1' and one for sex '2'. If the column is appropriate for the record, the value will read '1' and if not it will read '0', this will signify to the network in a binary manner the difference between the two. A similar issue can be seen for 'EDUCATION' and 'MARRIAGE' fields and as such a similar OHE will be required for these columns. Pandas has an inbuilt method 'get_dummies' which I will make use of for OHE.

B. Model Design

Design of the model will begin by building an Artificial Neural Network (ANN) with Sequential architecture. Being Sequential means that data will be passed between layers in order, this is suitable to the given dataset due to it being a Binary Classification problem. Other possible network configurations include Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN), these are deemed inappropriate for the dataset provided. CNNs are tailored for image processing and, whilst the data could be manipulated to present as an image, this is likely not optimal. RNNs are tailored for Natural Language Processing which again isn't the problem my model is tackling hence, again, an inappropriate network choice.

In regard to model layers, adding multiple Dense layers with ReLU activation is a good place to begin. The final output layer will have a 'Sigmoid' activation since the data output will be in the range [0, 1], namely either 0 or 1. From this point I can implement more or less layers based on test results.

To mitigate overfitting and improve performance, I will test adding Dropout layers in-between the Dense layers of the model. By adding drop out, data is intentionally lost between layers meaning less processing through the model and less chance of the model over-familiarising with the dataset. I will evaluate the merits of Dropout layers against the potential loss of accuracy, thus informing its appropriateness in a data-driven manner.

Keras handily comes with built in optimisers which will be applied to my model to automatically apply improvements between epochs. Being the most popular, I will test using the 'Adam', 'RMSprop' and 'SGD' optimisers, with preference toward 'SGD' due to it's efficient computation over large datasets. The loss function I will be using is 'Binary Cross Entropy', due to the binary nature of the output data, thus making it extremely applicable to my model.

Hyper-parameter tuning is the final step in my model design, by using a Grid Search algorithm, I can test building my model with a range of hyper-parameters to discover the most effective pairings of parameters. The Sklearn library packages the 'GridSearchCV' class, which will be used to perform a grid search on my model with cross-validation for more accurate validation metrics.

C. Validation Methods

K-Fold Cross-Validation is a widely used validation method: data is split into K groups, the model is trained using K-1 of these groups and then the K^{th} group is used for testing the model. By doing this, the risk of overfitting is mitigated by exposing the model to a range of data; a more reliable assessment of the reliability of the model is also discovered. This method of validation greatly increases computation for the program however it is a reasonable compromise to increase validation reliability. Sklearn provides the 'StratifiedKFold' class which I will use for the validation of my model. By using the stratified variant of a K-Fold class, it can be ensured that the proportion of features of interest is equal across folds, thus meaning no classes are over or under represented per fold.

In addition to Cross-Validation, I will graphically plot the model training and validation accuracy/loss which will visually highlight any overfitting problems.

IV. Experimental Results

Model implementation began by ensuring all appropriate classes were imported to the program, from here data was imported and for the first model I did not apply any pre-processing to see what results raw data achieved. A simple model containing three Dense layers yielded an accuracy score of 0.75, from here I began applying pre-processing steps.

OHE was applied to only the 'SEX' column of the data where I observed an accuracy reduction to 0.61 before including 'MARRIAGE' and 'EDUCATION' OHE yielding a new accuracy of 0.65. At the moment this is understandable due to the data not being normalised, applying the Scaler brought accuracy to a vastly improved 0.81. Applying some over or under-sampling to balance out the classes is a reasonable next step, it is important that this is done after data is split between train and test so no synthetic data is used for validation. Firstly, I used RUS which found an accuracy of 0.80, contrasting a 0.81 accuracy for ROS. SMOTE yielded a poor 0.39 accuracy

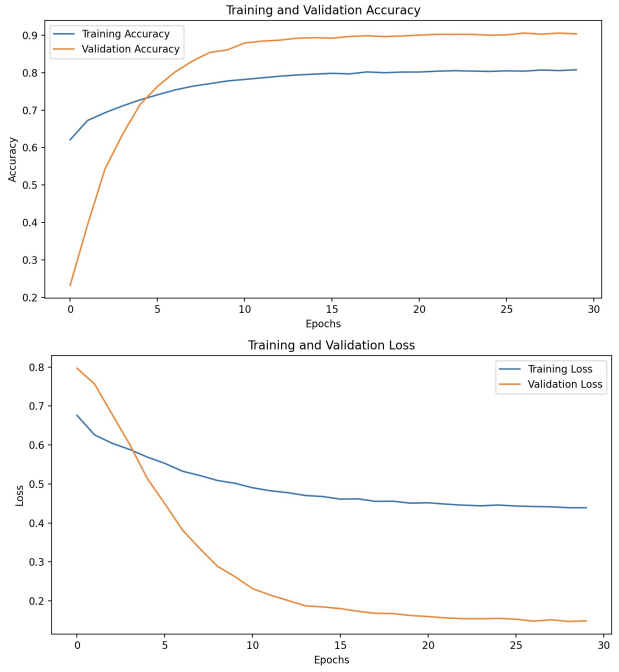

Fig. 1. SMOTE Validation Graphs

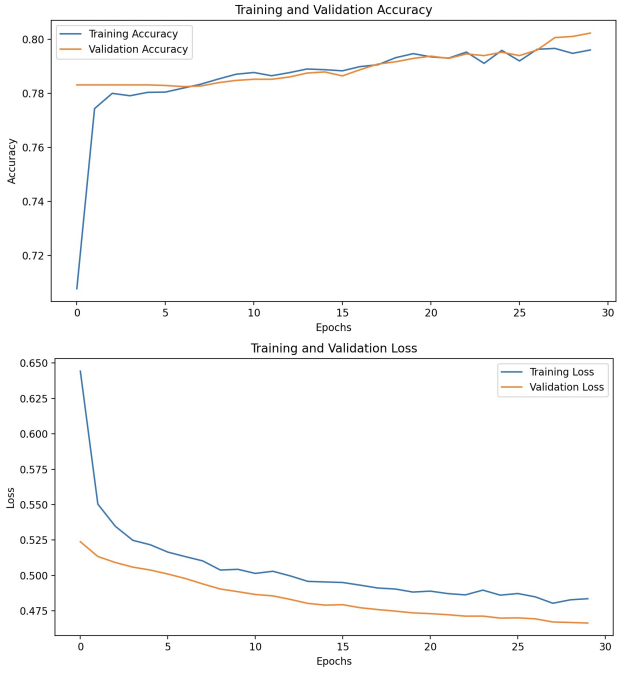

Fig. 2. No SMOTE Validation Graphs

however all three showed serious overfitting issues, as demonstrated in Figure 1 by the validation graphs from SMOTE. Due to this the decision was made to not include any over or under-sampling for the pre-processing stage, graphs are shown in Figure 2 after SMOTE was taken away demonstrating the difference.

Now appropriate pre-processing has been handled, I made

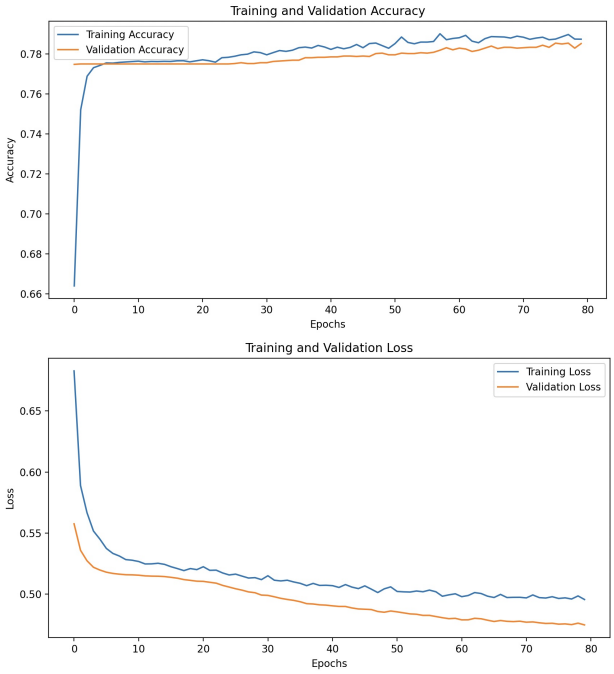

Fig. 3. Final Validation Graphs

changes to the model's architecture; systematically adding Dropout layers in order to mitigate overfitting risk and then changing the number of layers inside the model to find three Dense layers was optimal, again for overfitting mitigation. At times during testing an accuracy of 0.779 came up, this was a cause for concern since this number signifies the proportion of '0' results in the dataset, hence the model was always predicting '0'; more parameter tuning took place until the problem was resolved.

After testing the outlined optimisers I found that both 'Adam' and 'RMSprop' not only reduced accuracy but efficiency too, thus validating 'SGD' as the appropriate choice, as suspected.

At this point my model is ready for Grid Searching, I ran it with a parameter dictionary containing $epochs = [40, 60, 80]$, $batch\_size = [60, 90, 120]$ and $validation\_split = [0.2, 0.3, 0.4]$. Before running this I had manually tuned my dropout rate and neuron parameters to give a more stable configuration.

After Grid Searching to find an optimal configuration, the final model yielded a 79.5% accuracy and 0.463 loss score, with validation graphs shown in Figure 3 demonstrating a strong model with little overfitting. For more concrete validation I used a K-fold Cross-Validation with $K = 5$; finding an average validation accuracy of 78.74% for my final model, again validation plots are shown in Figure 4 to demonstrate how the model performs under Cross-Validation tests.

V. Summary

To summarise, by using a Sequential model ANN, with tuned hyper-parameters, a Credit Card user defaulting on their

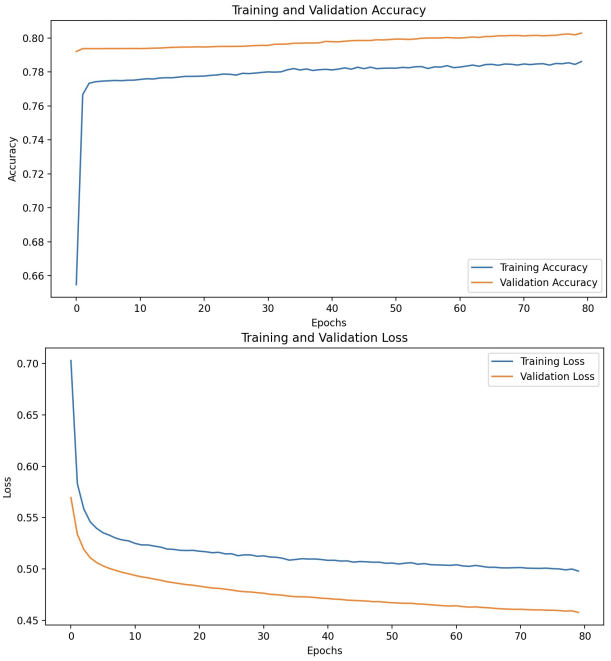

Fig. 4. K-Fold Cross Validation Graphs

payment can be predicted to a 78.7% accuracy and a loss of just 0.479. A 5-Fold Cross-Validation process has been undertaken to validate the rationality of the accuracy and loss values.

This report demonstrates how ANNs are suitable for Deep Learning on financial data; demonstrating how Deep Learning could be applied to other financial problems in order to reliably predict outcomes from data.

REFERENCES

[1] S. Hostetler, "Defaulting on credit cards: Why it happens and what to do." https://time.com/personal-finance/article/credit-card-default, 2024.

[2] M. Group, "Credit card fraud detection dataset." https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud, 2019.

[3] K. Team, "Keras documentation: Imbalanced classification: Credit card fraud detection." https://keras.io/examples/structured_data/imbalanced_classification/, 2020.

[4] B. Zhang, S. Li, and C. Yin, "A classification approach of neural networks for credit card default detection," *DEStech Transactions on Computer Science and Engineering*, 2017.

[5] H. He and Y. Ma, "Imbalanced learning: foundations, algorithms, and applications," 2013.

[6] S. Bhanja and A. Das, "Impact of data normalization on deep neural network for time series forecasting," *arXiv preprint arXiv:1812.05519*, 2018.