ECM2423: Deep Learning To Predict Credit Card Defaults

Joshua Tomsett
Department of Computer Science
University of Exeter
Exeter, UK
jt796@exeter.ac.uk

***Abstract*—This report address the challenge of credit card default predication with the use of deep learning techniques. It elaborates on the pre-processing of data, which includes normalization and managing the class imbalance by utilisation of SMOTENC. The architecture of the model is then detailed, describing its inputs, hidden layers, and the output layer. This is followed by an explanation of the approach to hyperparameter tuning and the rationale behind the selection of the specific loss function and activation function for the output layer. The performance of the model is illustrated through validation graphs and a table that documents the scores for various metrics: a loss score of 0.48, accuracy of 79.7%, precision of 54.8%, recall of 51%, and an area under the curve (AUC) score of 0.77. The report is summarized by highlighting the main findings, and a brief discussion on the future direction of the problem.**

I. Introduction

In the field of risk management, default risk, defined as the probability that cardholders will fail to make required payments on time, is vital. It requires comprehensive analysis of a wide array of variables and is therefore very suitable for a computational approach. Methods such as Logistics Regression [1], Decision Trees [2], and Neural Networks [3] have previously been employed to try and predict defaults. This report focuses on the latter method utilising a deep learning model and a large dataset of 30,000 records.

In order to guide this project I think it would be beneficial to consider how the model would actually be applied by financial institutions. Of course, their objective is to provided credit to all those who will pay it back, and deny those who will not, so in assessing the model's performance we must also assess the cost of denying credit against the cost of a default. This data is not publicly available and will change depending on the lender so my analysis will only be an estimate. Data from 2017 on Americas top credit card lenders revealed that on average each account generates $180 a year for the lender [4], and in 2023 average card debt increased by $3,235 [5]. In conjunction with the Federal Trade Commission finding that debt collectors buy $100 in debt from a bank for only $4 [6], we can estimate that the average default results in a loss of $3105. To reiterate, this is a very rough estimate, however, it can facilitate an understanding of the trade-off between detecting a default against wrongly denying credit. It follows that we can estimate the cost of a default to be about 17 time higher than the cost of denying credit. The metrics we use to evaluate success should reflect this; we aim to identify as many defaults as possible (i.e., maximize recall) while avoiding the incorrect denial of credit (i.e., maintain high precision).

Having highlighted the problem, we now turn our attention to the outline and achievements of the report. Initially we will discuss the pre-processing phase. Subsequently, the model design will be explained, followed by the obtained results. To conclude, a summary will encase the key findings.

II. Proposed Method

A. Pre-Processing Phase

In addition to normalization and resampling detailed subsequently, it is crucial to augment specific categorical features and appropriately split the dataset into training, validation, and testing segments. The 'PAY' values are adjusted from a range of [-2, 8] to [0, 10], and the 'SEX' values from [1, 2] to [0, 1]. This adjustment facilitates the use of embedding layers for categorical features, which require numerical inputs starting from 0 for efficient lookup operations. The data is split into 70% training, 15% validation, and 15% testing.

1) Normalization And Embedding: Currently, many features are on different scales and distributions, resulting in some values being significantly greater than others. To prevent undue influence of the model I employ normalization to the continuous features. This process standardizes all features to the same scale, achieving a mean of zero and a standard deviation of one, utilizing the Scikit-learn preprocessing module. For the categorical features I pass them through a embedding layer, turning the integers into dense vectors.

2) Class Imbalance: Among the 18000 data samples in the training set, 13993 belong to class 0 (non default) and 4007 belong to class 1 (default), establishing a 78:22 ratio. To address the class imbalance, the data must be resampled. Under-sampling, which entails the removal of samples to equalize class distribution, is generally discouraged due to the loss of valuable information. This assertion is supported by my own findings, leading to a preference for over-sampling. The particular technique chosen is SMOTENC, a variant of the Synthetic Minority Over-sampling Technique (SMOTE) for datasets containing both numerical and categorical features. It generates synthetic samples by drawing a line between existing samples, generating a new sample at a point along that line. By implementing SMOTENC with a sampling strategy

of 0.8, the minority class (class 1) was augmented to 80% of the majority class's size, increasing the minority class size to 11,194 samples. It is crucial to apply SMOTENC exclusively to the training dataset to prevent data leakage into the validation or testing sets.

B. The Model

1) Input Layer: The model comprises seven inputs: static continuous (LIMIT_BAL, AGE), SEX, EDUCATION, MARRIAGE, PAY, BILL_AMT, and PAY_AMT. The static inputs are categorized into a single input for continuous values and separate inputs for each categorical variable, as the latter must first be processed through an embedding layer. Upon embedding the categorical static features, all static variables are concatenated. The PAY input, encompassing the six categorical pay values, is also subjected to an embedding layer. Lastly, the inputs for BILL_AMT and PAY_AMT each contain six columns corresponding to their respective categories. After processing, these inputs result in four functional inputs: static, PAY, BILL_AMT, and PAY_AMT.

2) Hidden Layers: Each of the time series inputs (PAY, BILL_AMT, and PAY_AMT) traverses a LSTM layer with 64 neurons, followed by another LSTM layer with 32 neurons, both using the default tanh activation function. These LSTM layers learn and remember patterns over sequences of data, making it effective for tasks requiring understanding of time series data.

Following this, the outputs of the LSTM layers and concatenated with the static input. This concatenation then passes through three successive dense layers with 512, 256, and 64 neurons respectively. To facilitate a complex understanding of the combined data, the relu activation function is applied at each dense layer to introduce non-linearity. In an effort to mitigate overfitting and enhance the model's ability to generalize, L2 regularization is implemented across each dense layer, imposing penalties on large weights.

3) Output Layer: For the output layer we consider the nature of the problem, binary classification. To design a suitable output for this problem I employ a dense layer with a single neuron and the sigmoid activation function. This layer maps input values to a single value between 0 and 1, ensuring the output represents the probability of the input belonging to one of two classes. To quantify the performance of this model, binary cross-entropy is employed as the loss function, where the loss score increases as the predicted probability diverges from the actual label, guiding the model towards accurate predictions.

III. Experimental Results

A. Hyperparameter Settings

Having already stated the number of neurons, activation and loss functions used, I will now consider the other hyperparameters. To find the optimal configuration, I have employed a grid search using the following parameters:

- Batch size (32,64,128)
- Optimizer (Adam, RMSprop, SGD)
- Learning rate (0.001, 0.01, 0.1)
- Dense layer activation ('relu', 'leaky relu')

With 54 configurations evaluated, each was assessed based on its Area Under the Curve (AUC) score. I have chosen to use the AUC score for its ability to measure the model's effectiveness in distinguishing between the two classes, rather than relying only on raw accuracy.

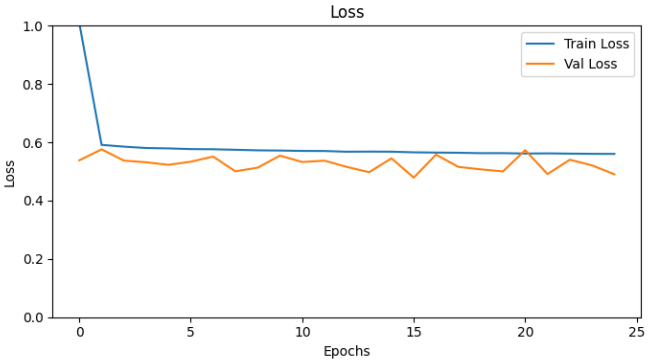
From the grid search I have found the optimal hyperparameters to be:

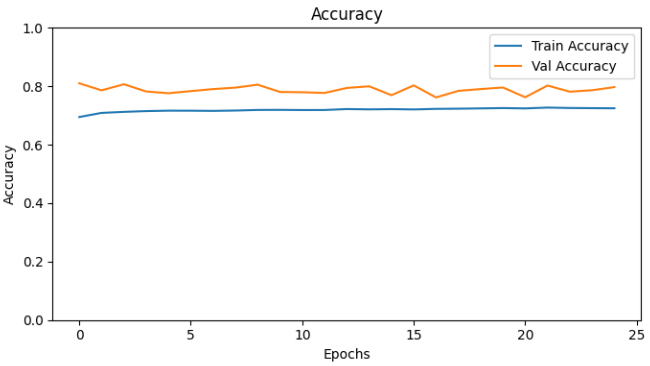
- Batch size = 64
- Adam optimizer with learning rate of 0.001
- Dense layers using 'relu' activation function

B. Obtained Results

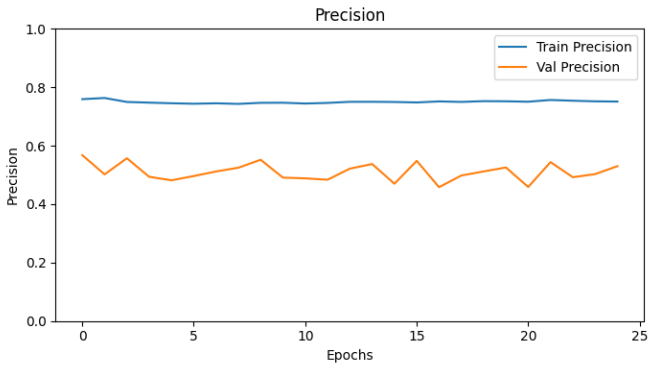
TABLE I
Test Set Metrics

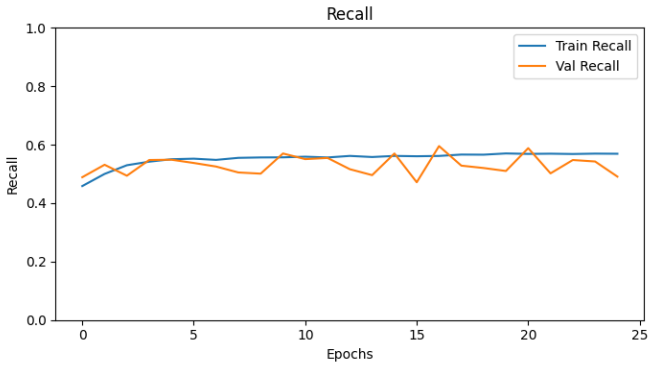
Metric	Value
Loss	0.48
Accuracy	0.797
Precision	0.548
Recall	0.51
AUC	0.77





After training the model with the optimal hyperparameters, you can observe the scores for each metric, as well as graphs depicting their change over training epochs.





With these graphs, we can asses not only how the model performs but also its ability to generalise to unseen data. From the loss graph, we observe a sharp initial drop which is common as the model begins to learn from the random initial weights. This is followed by rapid convergence, a trait observed across all metrics, which stabilize within 2 to 5 epochs. This likely stems from the utilization of the Adam optimizer, which dynamically adjusts its learning rate throughout the training process, frequently leading to swift convergence [7]. Except for precision, there is consistency in performance across all metrics between the training and validation sets. This consistency indicates that the model successfully avoids overfitting and exhibits strong generalization capabilities. However, a significant discrepancy is observed in precision between training and validation sets. This divergence might be attributed to the synthetic samples produced by SMOTENC. The alignment of test set outcomes with validation performance suggests that the issue lies with the training set, specifically that the training samples for the positive class are not representative of real world data. This suggest a potential issue with the synthetic data augmentation used, and that it does not accurately depict true instances of the positive class.

Having completed the analysis of the model's performance, we now refocus on the problem itself and the potential applications of the model by banks. As mentioned in the introduction, the aim is to identify as many defaults as possible, and to do so accurately. The test results indicate that the model can identify 51% of defaults and when it does predict a default it is correct around 55% of the time. Employing the previously calculated value—that a default is 17 times more costly than incorrectly denying credit—we can infer from the model's performance that its utilization will result in financial savings for the bank. For each 100 people we predict to default, 55 will do so.

IV. Summary

This report details a neural network built to predict credit card defaults. I have employed SMOTENC to address the class imbalance, and in doing so have achieved a model that is able to predict the majority of defaults. While acknowledging the potential for improvement, one must consider that the problem is challenging and can never to completed solved. The dataset comprises demographic information and credit histories of individuals; however, their payment behaviors are influenced by many external factors, including economic conditions and unforeseen large expenditures. Although an infallible model will always elude us, even an imperfect model, such as the one in this report, can provide great utility to those who have use for it.

References

[1] I.-C. Yeh and C.-h. Lien, "The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients," *Expert systems with applications*, vol. 36, no. 2, pp. 2473–2480, 2009.

[2] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, pp. 81–106, 1986.

[3] Y. Chen and R. Zhang, "Research on credit card default prediction based on k-means smote and bp neural network," *Complexity*, vol. 2021, pp. 1–13, 2021.

[4] J. Resendiz, "How much do credit card companies make per user?," 2017. Accessed: 10/03/2024.

[5] E. E. Issa, "2023 american household credit card debt study," 2024. Accessed: 10/03/2024.

[6] Federal Trade Commission, "The structure and practices of the debt buying industry." https://www.ftc.gov/sites/default/files/documents/reports/structure-and-practices-debt-buying-industry/debtbuyingreport.pdf, Jan. 2013. Accessed: 10/03/2024.

[7] R. Zaheer and H. Shaziya, "A study of the optimization algorithms in deep learning," in *2019 third international conference on inventive systems and control (ICISC)*, pp. 536–539, IEEE, 2019.