Developing a Forward Feed Neural Network to Predict Credit Card Defaults

Storm Prince

March 17, 2024

Abstract

To address class imbalance in the credit card default dataset, we found generating synthetic data for the minority class using SMOTE most effective in improving accuracy, allowing the neural network to learn the underlying patterns of the account that defaulted.

A sequential neural network with optimizer RMSprop and activation function Swish maximised the neural network's precision, ensuring the model was highly accurate in predicting actual defaults.

Our model generalised well to unseen data, making it applicable in real-world scenarios, but our design choice led to a low recall, which could cause financial consequences for credit card companies.

Our findings point towards a different approach, such as using a recurrent neural network to enhance the ability of the model to predict credit card defaults.

Our findings suggest that a different deep learning approach may be required to better capture the temporal sequences associated with our data.

1 Introduction

Credit card default occurs when the borrower fails to adhere to the repayment terms and conditions agreed upon with their credit card issuer, leading to several consequences such as late fees, high interest rates, and adverse effects on the individual's credit score. Financial institutions rely on earnings from interest and fees; defaulting on loans causes lost revenue and potential debt write-offs[1].

The objective of this report is to develop a deep learning model capable of generalising to unseen data to predict whether a customer will default on their credit card, which is a binary classification problem. It involves analysing a large array of features and extracting underlying meaningful patterns from highly dimensional data. Deep learning models excel at these tasks compared to traditional machine leading and statistical methods, producing more precise and reliable results.

Feedforward neural networks (FNN) are neural networks where data flows in one direction[2]. FNNs have been shown to be promising models in various predictive modelling tasks, including binary classification[4], making them a suitable candidate for predicting credit card defaults. They're able to capture complex information and are relatively simple to implement. Recurrent neural networks (RNN) offer a valuable approach for dealing with sequential or time-series data. RNNs consider information across previous inputs by sharing parameters across previous layers of the model[6]. Our problem includes analysing billing and payment trends overtime, suggesting that RNNs may perform better at capturing these temporal sequences, potentially improving the precision of credit card default predictions.

Our report is structured as follows: Section 2 introduces our model architecture, preprocessing phase and methodology. Section 3 presents findings from model testing and evaluates model effectiveness and generalisation capability. Section 4 provides summary of our selected methods, outcomes and practical implications.

2 Method

Our chosen approach utilises a feedforward neural network (FNN), where data is sequentially directed through the input layer to hidden layers, then finally to the output layer[2]. See Figure (1). We choose this method as they're relatively simple to implement, have been proven to perform well for binary classification tasks[4] and are suited to model complex tasks.

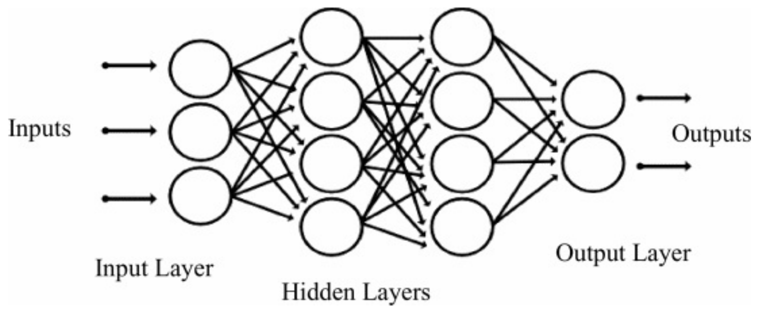

Figure 1: Visual representation of Feedforward Neutral Network architecture[3].

2.1 Initial preprocessing

```python
# Imports 'CCD.xls'.
CCD_import = pd.read_excel('CCD.xls', header=1)

# Preserves original dataset and removes duplicates
CCD_raw = (CCD_import.copy()).drop_duplicates()
```

We began by loading the credit card default (CDD) dataset, copying it to preserve the original then removing the duplicates. The rational behind this is that this data wasn't meant to be in the database and could skew performance.

```python
# One hot encoding Categoriacal data to improve deep learning performance.
CCD = pd.get_dummies(CCD_raw, columns=["SEX", "EDUCATION", "MARRIAGE"]).astype(int)
```

Our dataset contains three categorical variables, sex education and marriage. Each had low cardinality so we opted to used a standard approach using the get dummies function to one-hot encode these features. This expands the feature space into a binary representation, ensuring our model doesn't form ordinal relationships leading to improved machine learning performance.

```python
train_set, test_set = train_test_split(data, test_size =0.2, random_state = 42)
train_set, val_set = train_test_split(train_set, test_size=0.25, random_state = 42)
```

We divided our data into three subsets: training, validation, and testing. The training set is used during the training process to adjust feature weights. The validation set is used during training for hyperparameter tuning, facilitating monitoring of the model's performance on unseen data to correct underfitting and overfitting. The test set is the final untouched subset, used to evaluate the model's performance once training, validation, and model architecture are finalised. We used a 60%-20%-20% split of our data for the training, validation, and testing sets, respectively.

```python
def array_normalise(train, val, test):
  # Converts train, validation and test data into numpy arrays.
  train_features = np.array(train)
  val_features = np.array(val)
  test_features = np.array(test)
  # Normalises test, validation and test sets.
  scaler = StandardScaler()
  train_features = scaler.fit_transform(train_features)
  val_features = scaler.transform(val_features)
  test_features = scaler.transform(test_features)
  # Returns train validation and test features repsectively.
  return train_features, val_features, test_features
```

Our dataset is too variable to use directly and will lead to poor performance. To address this, we scale the data using StandardScalar, which is applied independently for each feature. This is a common requirement and necessary for machine learning applications, as they perform best when they look like standard, normally distributed data[7].

2.2 Overcoming Class Imbalance

The class imbalance in the CCD dataset after dropping duplicates is illustrated in Figure (2). This will lead our model to favour predicting the non-default class due to its predominance, thereby undermining its ability to accurately predict defaults. We will address this issue using two techniques.

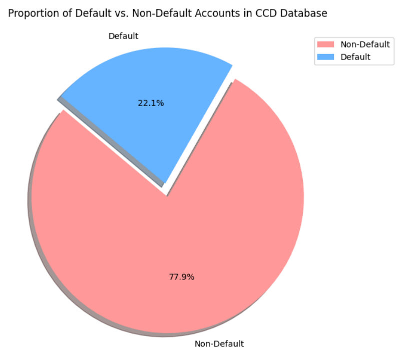

Figure 2: Distribution of default vs non-default in CCD.

```python
# Creates synthetic data for the minority class.
smote = SMOTE(random_state=42)
train_set, train_labels = smote.fit_resample(train_set, train_labels)
```

To mitigate the class imbalance, we implement the synthetic minority oversampling technique (SMOTE). SMOTE generates synthetic samples for the minority class for the training set, resulting in the model training on an imbalanced dataset. This approach aims to mitigate the model's bias towards the majority class, thereby improving its ability to generalise to unseen data as the model learns to recognise the minority class more effectively, leading to a more balanced performance across the classes[8].

```python
# Gets size of class
count_class_0, count_class_1 = train_set['default payment next month'].value_counts()

# Seperates defaults and non-defaults
CCD_class_0 = train_set[train_set['default payment next month'] == 0]
CCD_class_1 = train_set[train_set['default payment next month'] == 1]

# Randomly samples the majority class to have identical number of samples.
CCD_class_0_under = CCD_class_0.sample(count_class_1, random_state=42)

# Concatenate the undersampled class 0 DataFrame with the class 1 DataFrame
train_under = pd.concat([CCD_class_0_under, CCD_class_1], axis=0)

# Shuffles the train undersampled dataset.
train_under = train_under.sample(frac=1, random_state=42).reset_index(drop=True)
```

Our second method to mitigate class imbalance is undersampling. This involves randomly selecting samples from the majority class to balance the data set; the aim is the same as with SMOTE, but it has the drawback of potentially losing information.

2.3 Proposed Method

Our initial goal is to conduct an in-depth analysis of the best optimiser for predicting credit card defaults, focusing on Adam and RMSprop, for which we will explore which activation functions ReLU and Tanh or Swish offer the best performance and identify which class imbalance correction technique performs the best.

To achieve this, we will leverage the Hyperband tuner in Keras Tuner for hyperparameter optimization. This is a strategic approach to enhancing our feed-forward neural network by randomly iterating through various hyperparameter combinations and selecting the best-performing configurations based on validation set performance[9]. Which should allow us to determine the optimal combination of optimizer, activation function, and class imbalance correction technique. Using this method is particularly powerful as it assures that the model isn't being significantly hindered by incorrect hyperparameter settings when evaluating our optimizers and activation functions. Due to the complexity of this approach, we've included model, history and hyperparameter files.

To evaluate our model's performance, we will consider a variety of metrics. The list of metrics and representations in credit card default prediction is presented below:

Table 1: Metric evaluation

Metric	**Model Impact**
Accuracy	Percentage of correctly classified examples. Due to the inherent class imbalance, this metric is misleading and isn't a suitable measure to ensure generalisation of our model, as a model that just predicts the majority class will have an accuracy of approximately 78% (Figure 2) on the test set.
Recall	Percentage of actual defaults that were correctly classified. This is a useful metric to evaluate our model's performance, as it indicates its ability to accurately predict defaults.
Precision	The percentage of predictive defaults that were correctly classified provides insights on how well the model is predicting the minority class (defaults).
F1 Score	Harmonic mean of precision and recall. Due to class imbalance, it's a suitable metric for evaluating model performance and serves as an indicator of the model's effectiveness in predicting the minority class (defaults).

Prioritising recall would result in the highest number of credit card default predictions, but at a cost: decreased precision. This could potentially damage customer relationships and brand reputation.

3 Experimental results

For all models, we chose a batch size of 128, ran various models varying in batch size, and found minimal impact on performance. A larger batch size was chosen to reduce computational expense.
We manually tuned the class weights for the SMOTE and under-sampled training sets but didn't see a higher precision or recall. Although, the orignal training sets did exhibit better performance and we incorporated these by using automatic class weight selection.
While running the Hyperband tuner, we allow for adjustments in model architecture and hyperparameter configurations. Our design choices and the impact of these are summarised below:

Table 2: Model architecture and hyperparameters effect on model performance

Model feature	**Effect on model performance**
Dense layers: 1-5 Neurons: 10-150	Allows for modulation in model's complexity and learning capability.
Dropout: 0-0.5	Reduces over-fitting by constraining the system to enhance generalisation of model.
Learning Rate 0.001-0.00001	Identifies the optimal magnitude of the step size, enabling the selection which converges to global minimum, rather than being trapped in local minima to enhance learning efficiency and predictive accuracy[11].
Momentum (RMSprop): 0-0.9 (step 0.1)	Model will automatically determine the best momentum, mitigating the risk of overshooting the minimum[10].
Number of Epoches: 1-20	Number of times the model iterated over the entire training dataset. Allows for improved learning but with the penalty of a overfitting risk. To reduce this we will select the optimal epoch for the metric being optimised as well as leveraging Keras Earlystopping function to stop searching if the monitored metric stops improving no the validation set[5].
Class Weights	Used to address imbalances in traing datasets by applying highest weight to the minority class (defaults).

We began by determining the optimal activation function, optimal configuration, and validation metrics to maximise for our unaltered, SMOTE, and under-sampled training sets.

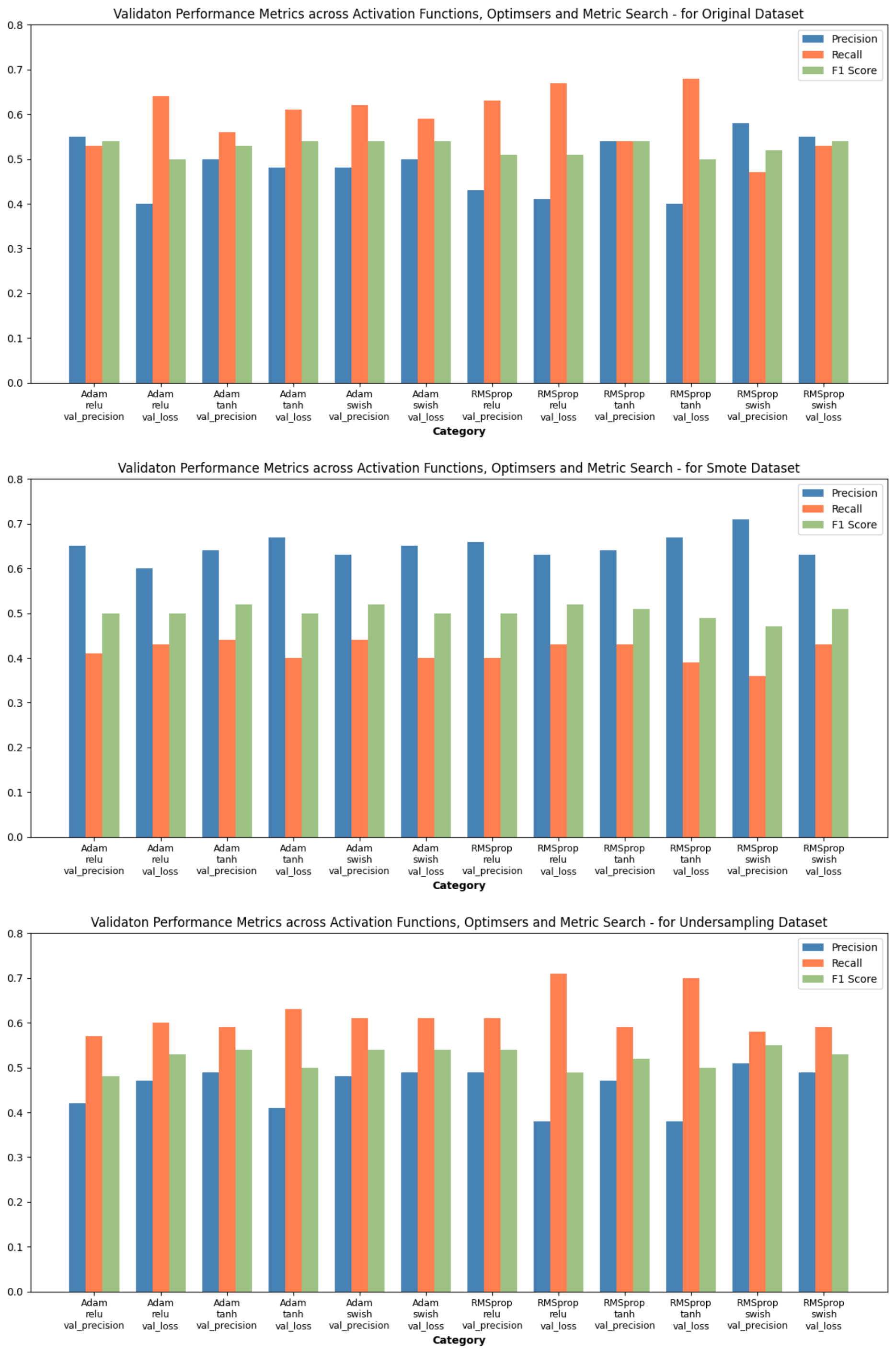

Figure 3: Graphs showing precision recall and F1 score on the validation set for the optimiser, activation function and metric being optimised for.

Optimal model configurations were chosen by maximising precision, presented below:

From Table (3), we choose SMOTE as our preprocessing method as it yielded the highest precision.

Preprocessing Method	Optimizer	Activation	Loss	Accuracy	Precision	Recall	F1
Original	RMSprop	swish	0.54	0.81	0.58	0.47	0.52
SMOTE	RMSprop	swish	0.45	0.82	0.71	0.36	0.47
Undersampling	RMSprop	swish	0.54	0.78	0.51	0.58	0.55

Table 3: Best performing configurations across preprocessing methods.

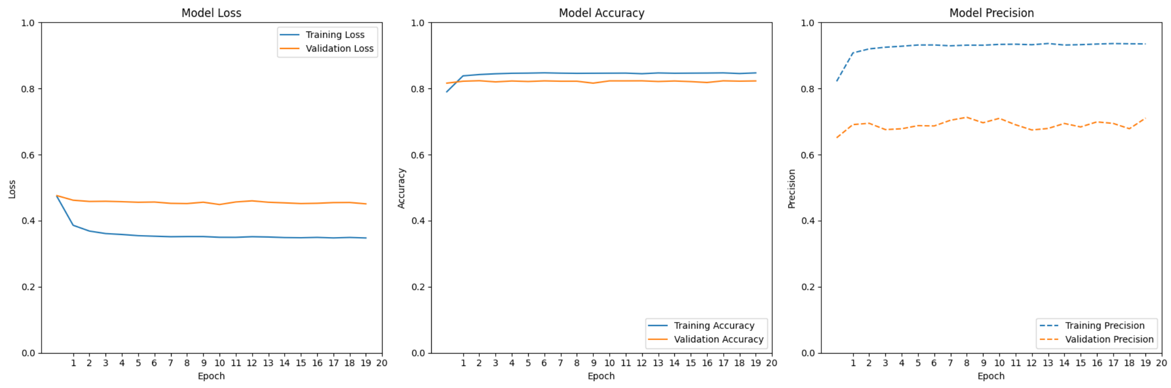

Figure 4: Training against validation loss, accuracy and precision against epochs.

Initially, in Figure (4), the training loss decreases, typical of the model fitting to the training data. The validation loss plateaus indicating that our model isn't overfitting or underfitting to the training data and should generalise well, further supported by the accuracy graphs of similar nature. We see small oscillations in the validation precision; however, the training precision plateaus. This further supports our notion that our model is not overfitting.

We then compared our validation data to the training data for our best model:

Data Type	Loss	Accuracy	Precision	Recall	F1
Validation:	0.45	0.82	0.71	0.36	0.47
Testing:	0.45	0.81	0.66	0.32	0.43

Table 4: Model performance for validation and testing sets.

Our model has similar loss, accuracy, and precision between the validation and testing sets; this suggests that our model has a strong ability to generalise and performs equally well on unseen data, making it useful in practical applications. It's able to predict the minority class (defaults) accurately 66% of the time but only captures 32% of the defaults, which means many of the defaults will go unrecognised, potentially causing financial consequences for the credit card company. Conversely, it has a relatively high prediction rate and may be suitable if its focus is to maintain customer relationships.

We incorporated a sequential mode where every dense layer except output has activation function Swish:

Dense(20) $\rightarrow$ Dropout 0.4 $\rightarrow$ Dense 140 $\rightarrow$ Dropout(0.3) $\rightarrow$ Dense(60) $\rightarrow$ Dense(130) $\rightarrow$ Dropout(0.4) $\rightarrow$ Dense(20) $\rightarrow$ Dropout(0.3) $\rightarrow$ Dense(1,'sigmoid')

Hyperparameters: batch size 128, epoch 9, learning rate 0.001, momentum 0.7.

4 Summary

The credit card default dataset presented a challenging task of dealing with class imbalance. We discovered that synthesising data for the minority class using SMOTE yielded the best precision and, therefore, the highest percentage of correctly identified defaults. Through various trials, we identified that the combination of RMSprop as the optimizer and Swish as the activation function maximised precision and found that our model didn't exhibit overfitting or underfitting. This resulted in our model generalising well to unseen data, indicating its applicability in practical applications.

However, our model exhibited a low recall, indicating that it overlooked a significant number of actual defaults. This could lead to financial repercussions for credit card companies by failing to identify defaulting customers. This was a deliberate design choice; we prioritised precision, which meant that when it predicts a default, it's likely to be correct, maintaining customer satisfaction by minimising false alarms.

References

[1] F. M. Talaat, Abdussalam Aljadani, M. Badawy, and Mostafa Elhosseini, “Toward interpretable credit scoring: integrating explainable artificial intelligence with deep learning for credit card default prediction,” Neural Computing and Applications, Dec. 2023, doi: https://doi.org/10.1007/s00521-023-09232-2. (accessed Mar. 13, 2024).

[2] W. A. Chishti and S. M. Awan, “Deep Neural Network a Step by Step Approach to Classify Credit Card Default Customer,” IEEE Xplore, Nov. 01, 2019. https://ieeexplore.ieee.org/document/8966723. (accessed Mar. 17, 2024).

[3] Mathworks, “What Is Deep Learning? — How It Works, Techniques Applications,” Mathworks.com, 2019. https://uk.mathworks.com/discovery/deep-learning.html. (accessed Mar. 15, 2024).

[4] S. hegde and M. R. Mundada, “Enhanced Deep Feed Forward Neural Network Model for the Customer Attrition Analysis in Banking Sector,” International Journal of Intelligent Systems and Applications, vol. 11, no. 7, pp. 10–19, Jul. 2019, doi: https://doi.org/10.5815/ijisa.2019.07.02. (accessed Mar. 15, 2024).

[5] “Introduction to the Keras Tuner — TensorFlow Core,” TensorFlow. https://www.tensorflow.org/tutorials/keras/keras_tuner. (accessed Mar. 15, 2024).

[6] IBM, “What are Recurrent Neural Networks? — IBM,” www.ibm.com, 2023. https://www.ibm.com/topics/recurrent-neural-networks. (accessed Mar. 15, 2024).

[7] Scikit-Learn, “sklearn.preprocessing.StandardScaler — scikit-learn 0.21.2 documentation,” Scikit-learn.org, 2019. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.(accessed Mar. 17, 2024).

[8] D. Elreedy and A. F. Atiya, “A Comprehensive Analysis of Synthetic Minority Oversampling Technique (SMOTE) for handling class imbalance,” Information Sciences, vol. 505, pp. 32–64, Dec. 2019, doi: https://doi.org/10.1016/j.ins.2019.07.070. (accessed Mar. 17, 2024).

[9] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, “Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization,” Journal of Machine Learning Research, vol. 18, no. 185, pp. 1–52, 2018, Available: https://jmlr.org/papers/v18/16-558.html.(accessed Mar. 17, 2024).

[10] F. Franco, “What is RMSprop? (with code!),” Medium, Oct. 27, 2023. https://medium.com/@francescofranco_39234/what-is-rmsprop-0f54effc47e4(accessed Mar. 17, 2024).

[11] I. Dabbura, “Gradient Descent Algorithm and Its Variants,” Medium, Sep. 27, 2022. https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3#:~:text=We%20can%20use%20fixed%20learning(accessedMar.17,2024). (accessed Mar. 17, 2024).