ECM2423 Coursework

Building a Deep Learning Model Using Credit Card Default Data

March 14, 2024

Abstract

This report delves into the development of a non-sequential, branch-based deep neural network aimed at predicting credit card defaults. Using a dataset of 30,000 Taiwanese customers, the network seeks to accurately classify whether a customer will default on their credit card payments. While various machine learning techniques exist for this task, the report focuses solely on a logistic regression-based approach due to the absence of necessary spatial and temporal features for other deep learning architectures. It details the model's design influences, bias mitigation techniques, data pre-processing methods, and the rationale behind selecting a branch-based structure. Different model architectures are explored, and hyperparameters are tuned to encourage generalization and prevent overfitting. The final model achieves a rounded accuracy score of 70%, demonstrating effective bias avoidance with an acceptable F1 score of 0.67. However, the report acknowledges the limitations of the biased dataset and suggests avenues for future improvement.

1 Introduction

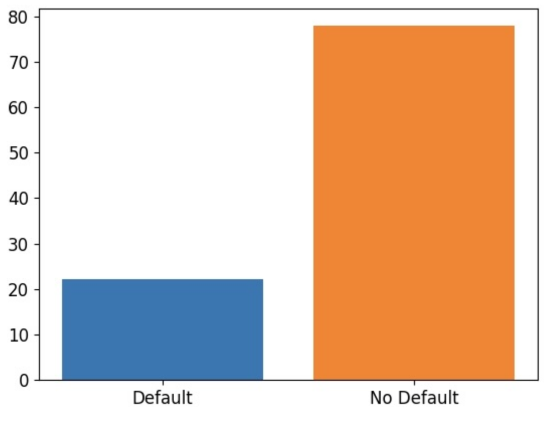
This report aims to set out the logic, decisions, and data which influenced the creation of a non-sequential, branch based, deep neural network for identifying credit card defaults.
The network aims to correctly predict whether a customer will default on their credit card payments using a real dataset of 30,000 Taiwanese customers [1]. The dataset includes personal information such as education level, sex, and marital status, as well as information about their credit card payments over the last 6 months, comprising of monthly billed amounts, paid amounts, and how late their payments were.
Currently, a variety of machine learning approaches are used to detect fraud and predict payment defaults. Some of the more common techniques include dynamic random forest, k-nearest neighbour, and decision trees [2][3]. Indeed, this exact dataset has been combined with live transactions in a model which takes advantage of temporal data, along with the standard feature set described above [4].
However, this report concentrates solely on a logistic regression-based approach. Specifically, the lack of spatial and temporal features makes convolution- and recurrence-based networks inappropriate for the assigned task. Instead, this project uses a fully connected network, with the aim of exploiting feature correlations through network branch with separate inputs.
The report will first explain the influences on the model's design, starting with the different bias mitigation techniques investigated and data pre-processing methods used, before analysing several different model architectures and explaining the motivations behind choosing a branch-based structure. The model's hyperparameter setting will be analysed, before evaluating the model's performance based on a variety of metrics. Finally, the findings will be summarised, and future research areas and improvements discussed.

2 Proposed Method

2.1 Bias

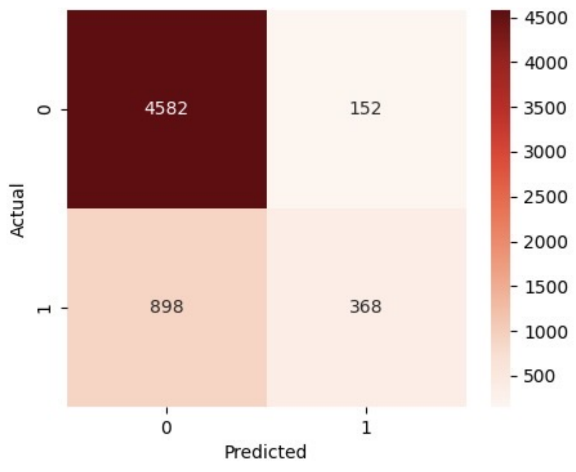
The most significant problem encountered throughout this exercise was that of data bias. The credit card default dataset provided has a built-in bias of 78% towards negative outcomes, meaning just 22% of the samples are defaults.

Figure 1: Probability Of Defaulting Payment Next Month



Correcting this bias is crucial, as not doing so allows the model to achieve an accuracy of 80% simply by predicting "no default" every time.

Figure 2: Confusion Matrix - No Bias Correction

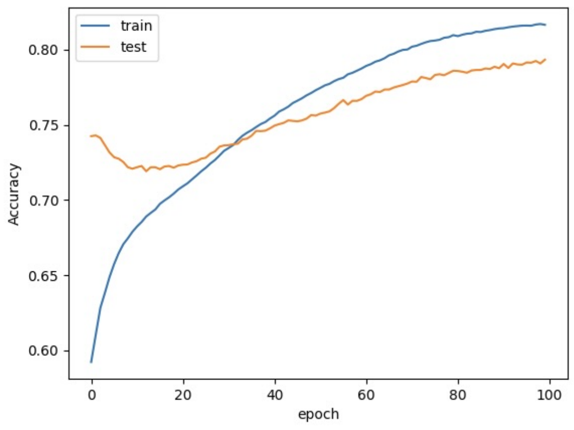


Three different strategies were tested to remove this bias: SMOTE, SMOTE and random under sampling, and random under sampling with k-fold evaluation.

2.1.1 SMOTE

Initially, SMOTE seemed the most well-suited procedure, as it corrected the bias without deleting samples, allowing the network to learn from as many real samples as possible. To evaluate SMOTE's effectiveness, 20% of the data was reserved for testing, while the remaining 80% was oversampled to remove bias and used to train a shallow network. The accuracy graph seemed positive; the initial drop in validation accuracy followed by a steady increase implied that the model was learning to distinguish samples.

Figure 3: Accuracy - SMOTE



However, test precision and recall were low despite high training precision and recall. This meant the model was learning to correctly identify more defaults in the training data than the testing data, which can be clearly seen when the network is tested on unbiased data. The model was learning to correctly identify the SMOTE generated defaults but not the real-life ones.

Dataset	Accuracy	Precision	Recall
SMOTE Training	0.8172	0.8686	0.7475
SMOTE Validation	0.7940	0.5133	0.4589
SMOTE Validation – Unbiased Test	0.6085	0.7571	0.3195

This discrepancy is likely caused by a combination of the sheer amount of oversampling taking place (roughly 80% of the default training samples are synthetic), along with known SMOTE issues

Figure 4: Accuracy - Undersampling and k-Fold

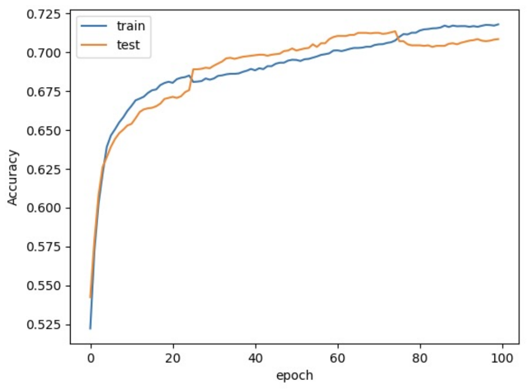
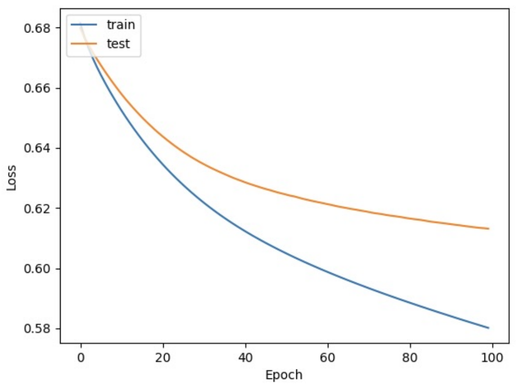


Figure 5: Loss - SMOTE and Random Undersampling



such as sample overlapping and noise interference which blur the true class boundary [5]. More finely tuned oversampling techniques may be able to produce synthetic data more effectively; ASN-SMOTE is able to avoid class overlap by also accounting for the nearest majority class samples [6]. However, such techniques are outside the scope of this report.

2.1.2 SMOTE and Random Undersampling

As previously stated, the fact that 80% of default training samples was synthetic likely caused overfitting to SMOTE generated data. Following this hypothesis, the next bias prevention technique created less synthetic data by using SMOTE with a lower oversampling rate, and corrected the bias further with random under sampling.
This technique produced a much fairer model with a testing recall of 0.5931. However, a SMOTE ratio of more than 0.35 resulted in overfitting to synthetic data as previously seen. Furthermore, accuracy dropped to 0.6704, which is likely due to high levels of under sampling.

2.1.3 Random Undersampling and k-Fold Evaluation

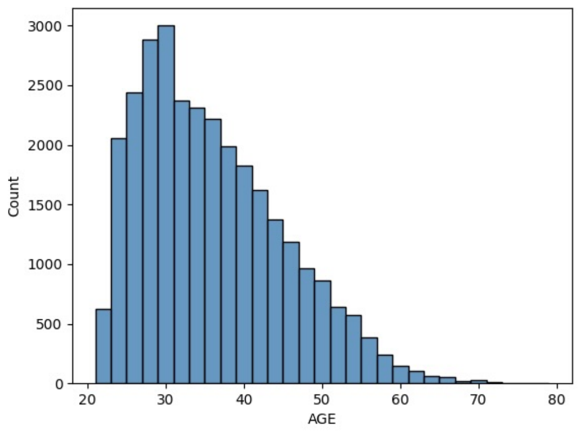
The main problem of the previous bias resolution method was the lack of samples, resulting in a significant drop in accuracy. One common method used with a small dataset is k-fold evaluation, where data is split into k segments, and each segment is used to train the model $k-1$ times and validate the model once.
This method facilitates the use of a smaller unseen validation set, so no oversampling was used. Despite this, the size of the training set was 4154 samples larger, as k-fold validation allowed the final unseen validation dataset to be half the previous size.
Using k-fold validation resulted in a final validation accuracy of 68%, with precision and recall of 0.70 and 0.62, a slight improvement on SMOTE with random undersampling. Critically, k-fold validation resulted in significantly less overfitting than the previous methods; the evaluation loss differed from the training loss by just 0.02.

For this reason, k-fold evaluation with undersampling was the chose bias removal technique.

2.2 Pre-Processing

The data pre-processing of the continuous features used z-score standardisation, which gave every continuous feature a mean of 0 and standard deviation 1. This strategy was picked to reduce the difference in scale of the continuous variables, and was deemed appropriate as the age feature already somewhat followed a Gaussian distribution.

Figure 6: Age Distribution



However, the balance limit, balance amounts, and payment amounts features all followed non-normal distributions with significant ranges. Therefore, log was applied to all these features, which transformed them into more appropriate distributions, while also improving the network's stability by removing extremities.

Figure 7: Bill Amount 1 Distribution

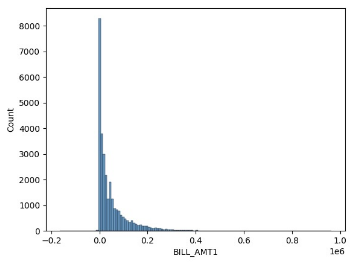


Figure 8: Limit Balance Distribution

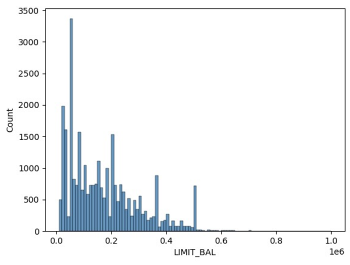
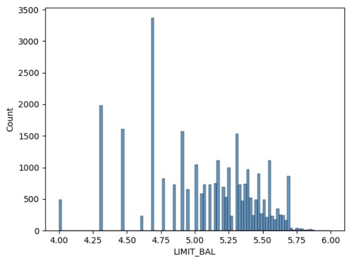


Figure 9: $\log_{10}$ Limit Balance Distribution



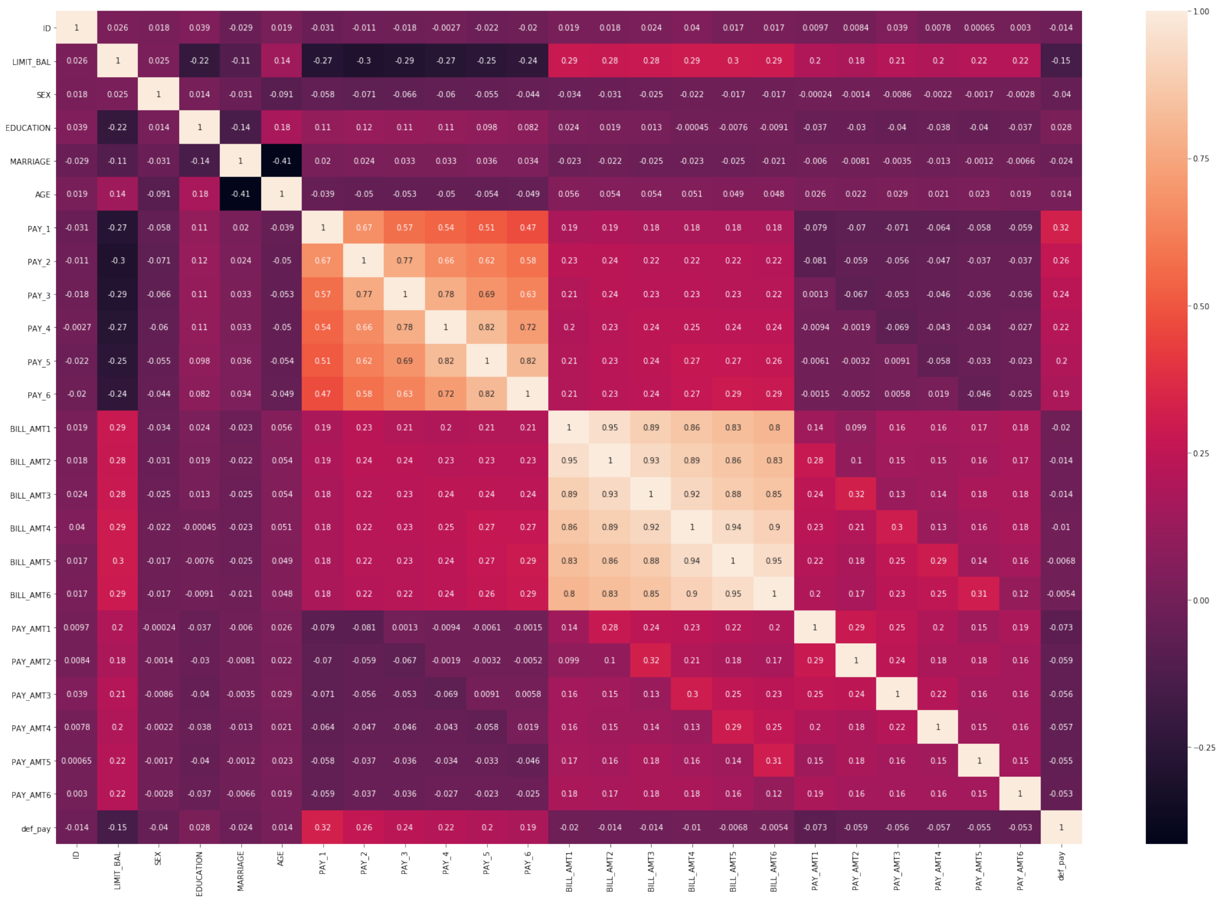
Finally, some of the bill amount features had sporadic negative values. These were deemed to be erroneous and detrimental to the network, and hence all bill amount features were clipped to be no less than 0.

All the non-continuous classification features, such as payment lateness, marital status, and sex, were One-Hot-Encoded. This was because One-Hot-Encoding would scale the classification inputs in line with the standardised continuous inputs, increasing model stability. Furthermore, one superfluous dimension was removed from each feature's One-Hot-Encoded vector to reduce unnecessary complexity and avoid the "curse of dimensionality".

2.3 Model Architecture

Graphing the correlation between different features reveals a connection between data related to transaction history. Specifically, data relating to a specific month's credit card bill correlates with the data from the month before. This is expected – each month's subset is temporally connected to the next.

Figure 10: Correlation Matrix



To take advantage of this in a model, branches were used, where each branch's input was a subset of correlated data. Each branch would then draw connections between correlated data and concentrate these links into a few key qualities.

The first model (see page 14) sought to take full advantage of the temporal aspect of the data by grouping every month's transaction data into separate inputs, with a separate input for non-transaction data. A set of layers would process one month's transaction data, then output to another set of layers which analysed the next month's transaction data, and so on. However, this model failed abjectly, as its significant depth made it extremely vulnerable to the vanishing gradient problem.
The second attempted model (see page 13) took the same inputs of the first, but combined them all at once instead of sequentially. This removed the problem of vanishing gradients, allowing the model to learn. However, this model suffered from overfitting, likely due to the significant number of inputs at the final concatenation layer allowing the model to find a perfect solution – the so-called "curse of dimensionality".
The final model (see page 12) sought to simplify the previous architectures and prevent excessive dimensionality by combining the branches into two: one for transaction data and one for personal data. This allowed for much wider branches and thus more complex connections to be found between correlated inputs.

3 Evaluation

3.1 Hyperparameters

3.1.1 Optimiser, Learning Rate, and Epochs

The overall goal of hyperparameter tuning was to encourage generalisation through rapid learning over a short period of time. Similar results could be achieved by the SGD, Adam, and RMSProp optimisers with appropriate learning rate. However, RMSProp tended to approach the targeted weights smoother than the others, so it was chosen.

Figure 11: Loss - Adam; learning rate = 0.00001

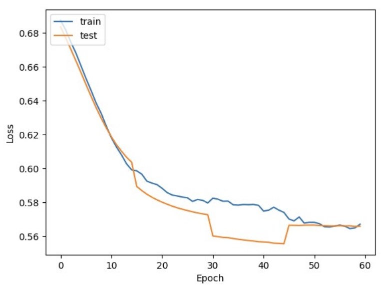


Figure 12: Loss - RMSProp; learning rate = 0.00001

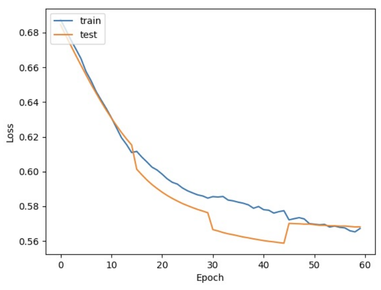
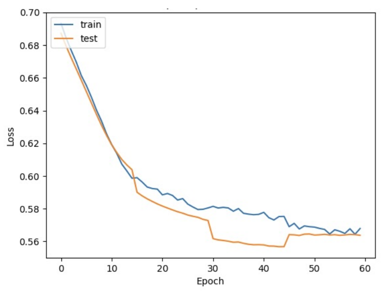


Figure 13: Loss - SGD; learning rate = 0.006



The learning rate was chosen to maximise the speed of the model's convergence while limiting the irregularity of the descent. A learning rate of 0.0001 allowed for 60 epochs while maintaining a relatively smooth loss graph.

3.1.2 Batch Size

Batch size was tuned to generalise the model as much as possible and discourage bias, while also being small enough to converge quickly. A batch size of 150 was chosen to achieve this, as batch sizes around 200 began to slow the model's convergence.

3.1.3 Loss Function

Two loss functions were looked at during hyperparameter tuning: binary cross-entropy and hinge. The hinge loss function is defined as:

$$\mathscr{L}(y) = \max(1, 1 - y \cdot y) \tag{1}$$

where $t = \pm 1$ is the intended outcome, and y is the predicted outcome. This makes hinge is a linear loss function, while binary cross-entropy is logarithmic. As this model aims to prevent bias towards the negative outcome, binary-cross entropy was chosen, as a logarithmic loss would punish a bias more severely than a linear loss.

3.1.4 Dropout

Dropout was tuned to minimise under and over fitting, while also being as low as possible to maximise training efficiency. It was found experimentally that a dropout rate of 0.1 between every layers prevented overfitting while also facilitating rapid training.

Figure 14: Loss - dropout = 0.1

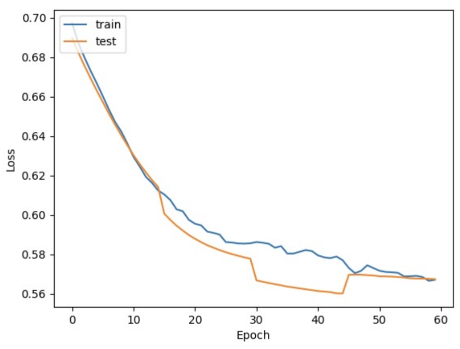
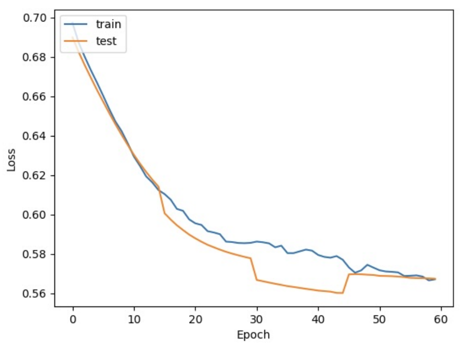


Figure 15: Loss - dropout = 0.2



3.1.5 Depth and Width

The aim of the branched based structure is to allow for correlations in each input to be found and exploited. As such the branches were made wide and relatively shallow, with the width decreasing down each branch. The width of the branches allows the network to extract complex connections between input features, while the tapering forces the model to extract only the most important features, and prevents over abstraction [7].

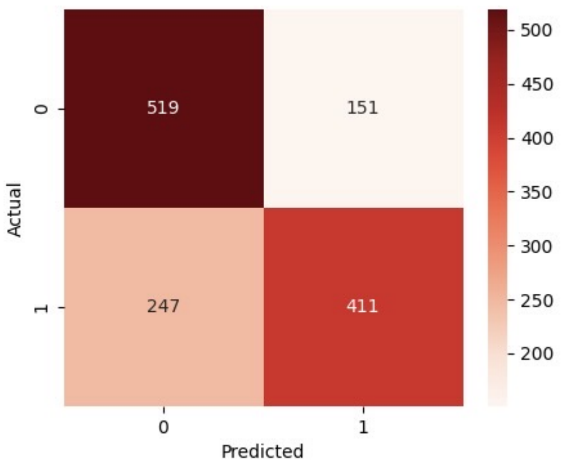
3.2 Results

The final model was run 100 times, and the following average metrics collected:

	Loss	Accuracy	F1 Score	Precision	Recall
Score	0.57547	0.69920	0.67111	0.73231	0.61949

As previously explained, the main challenge of this data set is the bias and relative difficulty of creating a model which successfully identifies defaults and well as non-defaults. The fact that the F1 score is high demonstrates that the network is working with limited bias, which is further supported by the confusion matrix and similar precision and recall scores.

Figure 16: Confusion Matrix - Final Model



While this logistic regression-based approach has proven to be fruitful, there is significant room for improvement, particularly in bias removal. The solution used for this network is sufficient but necessitates discarding 60% of the data. The use of a more sensitive oversampling method which facilitates the use of this data, such as ASN-SMOTE, could prove prosperous.
Furthermore, the use of a non-deep-learning-based models, such as random forests, has been shown to yield superior results [4]. Therefore, should further work take place, a more holistic, well-rounded analysis of different models' performance on this data is appropriate.

4 Summary

This report has provided an in-depth look into the design choices behind a branch based deep neural network for predicting credit card payment defaults. The problem of bias has been explored, and multiple different solutions, including k-fold evaluation, were discussed, and evaluated. A combination of logarithms, z-score standardisation, and one-hot-encoding were used to pre-process data in a network appropriate fashion. The correlation-based motivation behind a branch structured network was explained, and the different network iterations and their respective drawbacks were examined. Finally, the hyperparameters were analysed and tuned with network training speed, bias dissuasion, and over and under fitting in mind. The network produced had a rounded accuracy score of 70%, while demonstrating a sufficient bias avoidance through an acceptable F1 score of 0.67.

References

[1] UCI Machine Learning Repository, "Default of Credit Card Clients Dataset," https://www.kaggle.com/datasets/uciml/default-of-credit-card-clients-dataset, Insert Year, accessed on: Insert Date.

[2] S. Nami and M. Shajari, "Cost-sensitive payment card fraud detection based on dynamic random forest and k-nearest neighbors," *Expert Systems with Applications*, vol. 110, 06 2018.

[3] J. K. Afriyie, K. Tawiah, W. A. Pels, S. Addai-Henne, H. A. Dwamena, E. O. Owiredu, S. A. Ayeh, and J. Eshun, "A supervised machine learning algorithm for detecting and predicting fraud in credit card transactions," *Decision Analytics Journal*, vol. 6, p. 100163, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2772662223000036

[4] S. R. Islam, W. Eberle, and S. K. Ghafoor, "Credit default mining using combined machine learning and heuristic approach," *ArXiv*, vol. abs/1807.01176, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:49563351

[5] Z. Jiang, T. Pan, C. Zhang, and J. Yang, "A new oversampling method based on the classification contribution degree," *Symmetry*, vol. 13, no. 2, 2021. [Online]. Available: https://www.mdpi.com/2073-8994/13/2/194

[6] X. Yi, Y. Xu, Q. Hu, S. Krishnamoorthy, W. Li, and Z. Tang, "Asn-smote: a synthetic minority oversampling method with adaptive qualified synthesizer selection," *Complex Intell. Syst.*, vol. 8, p. 2247–2272, 2022.

[7] T. Nguyen, M. Raghu, and S. Kornblith, "Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=KJNcAkY8tY4

A Model Architecture Diagrams

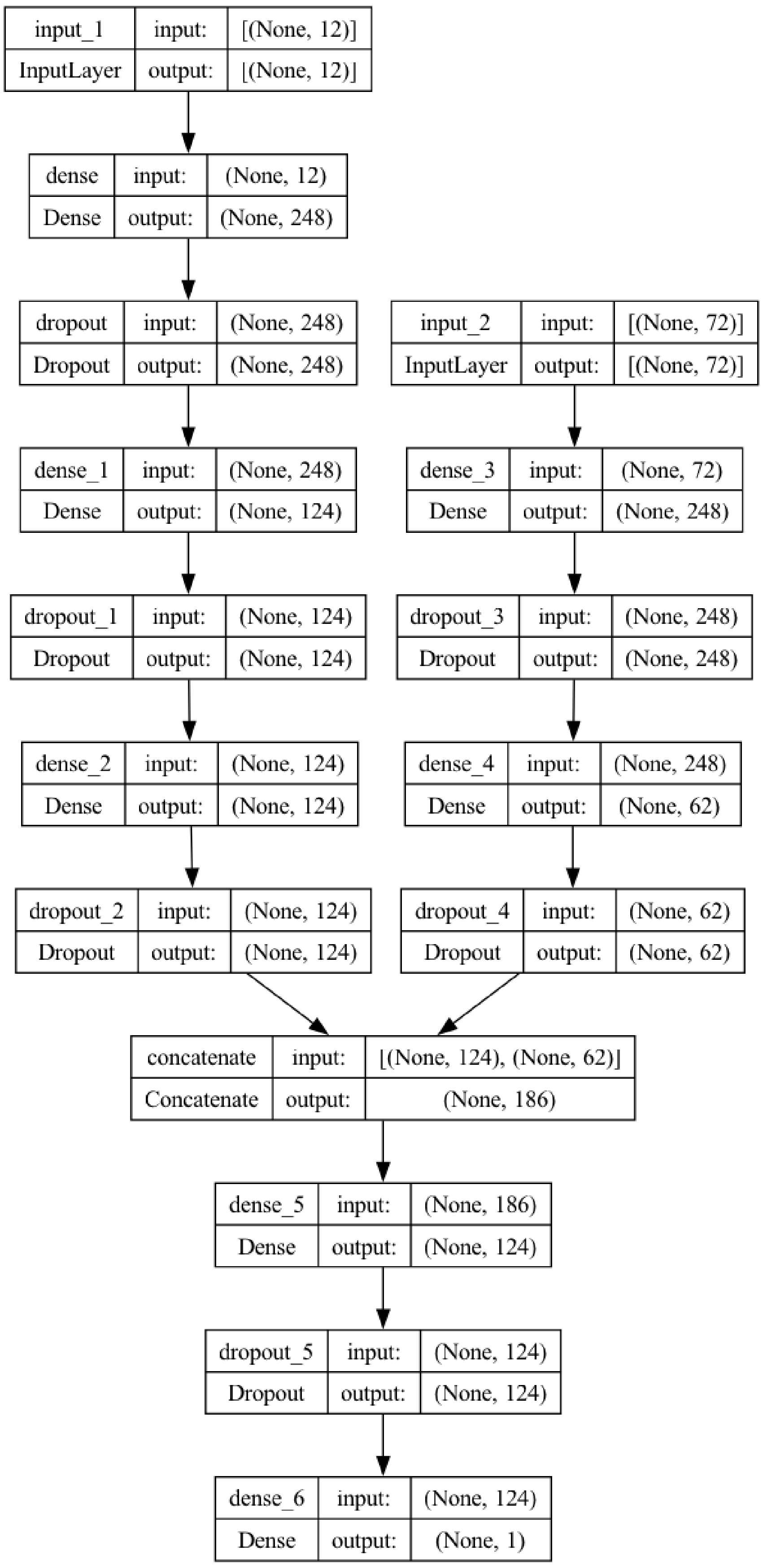

input_1 | input: | [(None, 12)]
InputLayer | output: | [(None, 12)]
dense | input: | (None, 12)
Dense | output: | (None, 248)
dropout | input: | (None, 248)
Dropout | output: | (None, 248)
dense_1 | input: | (None, 248)
Dense | output: | (None, 124)
dropout_1 | input: | (None, 124)
Dropout | output: | (None, 124)
dense_2 | input: | (None, 124)
Dense | output: | (None, 124)
dropout_2 | input: | (None, 124)
Dropout | output: | (None, 124)
input_2 | input: | [(None, 72)]
InputLayer | output: | [(None, 72)]
dense_3 | input: | (None, 72)
Dense | output: | (None, 248)
dropout_3 | input: | (None, 248)
Dropout | output: | (None, 248)
dense_4 | input: | (None, 248)
Dense | output: | (None, 62)
dropout_4 | input: | (None, 62)
Dropout | output: | (None, 62)
concatenate | input: | [(None, 124), (None, 62)]
Concatenate | output: | (None, 186)
dense_5 | input: | (None, 186)
Dense | output: | (None, 124)
dropout_5 | input: | (None, 124)
Dropout | output: | (None, 124)
dense_6 | input: | (None, 124)
Dense | output: | (None, 1)

Figure 17: Final Model Architecture

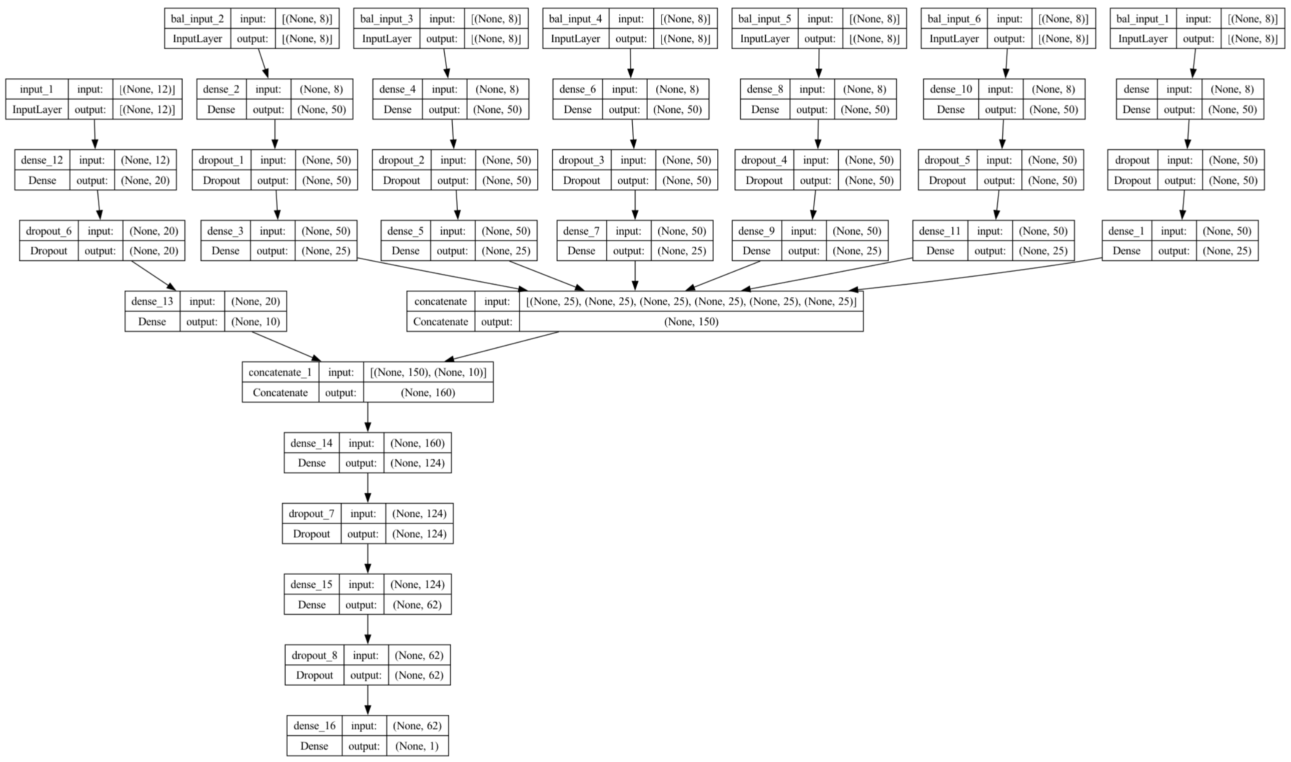

bal_input_2 | InputLayer | input: [(None, 8)] | output: [(None, 8)]
bal_input_3 | InputLayer | input: [(None, 8)] | output: [(None, 8)]
bal_input_4 | InputLayer | input: [(None, 8)] | output: [(None, 8)]
bal_input_5 | InputLayer | input: [(None, 8)] | output: [(None, 8)]
bal_input_6 | InputLayer | input: [(None, 8)] | output: [(None, 8)]
bal_input_1 | InputLayer | input: [(None, 8)] | output: [(None, 8)]
input_1 | InputLayer | input: [(None, 12)] | output: [(None, 12)]
dense_2 | Dense | input: (None, 8) | output: (None, 50)
dense_4 | Dense | input: (None, 8) | output: (None, 50)
dense_6 | Dense | input: (None, 8) | output: (None, 50)
dense_8 | Dense | input: (None, 8) | output: (None, 50)
dense_10 | Dense | input: (None, 8) | output: (None, 50)
dense | Dense | input: (None, 8) | output: (None, 50)
dense_12 | Dense | input: (None, 12) | output: (None, 20)
dropout_1 | Dropout | input: (None, 50) | output: (None, 50)
dropout_2 | Dropout | input: (None, 50) | output: (None, 50)
dropout_3 | Dropout | input: (None, 50) | output: (None, 50)
dropout_4 | Dropout | input: (None, 50) | output: (None, 50)
dropout_5 | Dropout | input: (None, 50) | output: (None, 50)
dropout | Dropout | input: (None, 50) | output: (None, 50)
dropout_6 | Dropout | input: (None, 20) | output: (None, 20)
dense_3 | Dense | input: (None, 50) | output: (None, 25)
dense_5 | Dense | input: (None, 50) | output: (None, 25)
dense_7 | Dense | input: (None, 50) | output: (None, 25)
dense_9 | Dense | input: (None, 50) | output: (None, 25)
dense_11 | Dense | input: (None, 50) | output: (None, 25)
dense_1 | Dense | input: (None, 50) | output: (None, 25)
dense_13 | Dense | input: (None, 20) | output: (None, 10)
concatenate | Concatenate | input: [(None, 25), (None, 25), (None, 25), (None, 25), (None, 25), (None, 25)] | output: (None, 150)
concatenate_1 | Concatenate | input: [(None, 150), (None, 10)] | output: (None, 160)
dense_14 | Dense | input: (None, 160) | output: (None, 124)
dropout_7 | Dropout | input: (None, 124) | output: (None, 124)
dense_15 | Dense | input: (None, 124) | output: (None, 62)
dropout_8 | Dropout | input: (None, 62) | output: (None, 62)
dense_16 | Dense | input: (None, 62) | output: (None, 1)

Figure 18: Non-Sequential Model Architecture

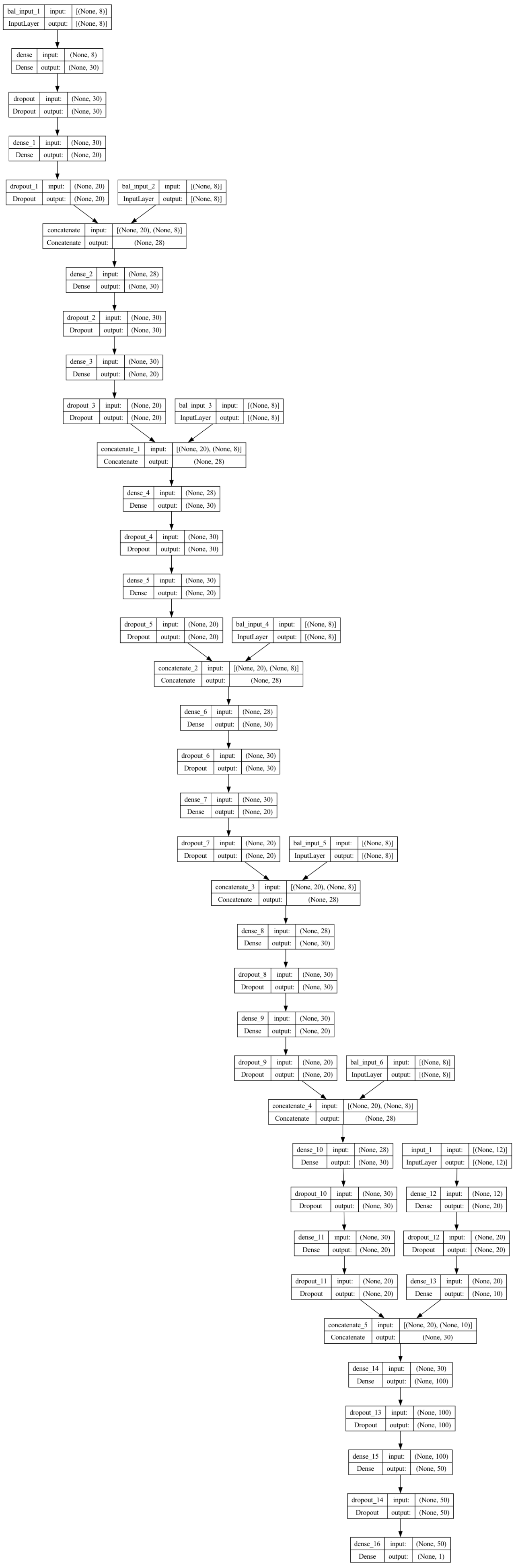

bal_input_1 | input: | [(None, 8)]
InputLayer | output: | [(None, 8)]
dense | input: | (None, 8)
Dense | output: | (None, 30)
dropout | input: | (None, 30)
Dropout | output: | (None, 30)
dense_1 | input: | (None, 30)
Dense | output: | (None, 20)
dropout_1 | input: | (None, 20)
Dropout | output: | (None, 20)
bal_input_2 | input: | [(None, 8)]
InputLayer | output: | [(None, 8)]
concatenate | input: | [(None, 20), (None, 8)]
Concatenate | output: | (None, 28)
dense_2 | input: | (None, 28)
Dense | output: | (None, 30)
dropout_2 | input: | (None, 30)
Dropout | output: | (None, 30)
dense_3 | input: | (None, 30)
Dense | output: | (None, 20)
dropout_3 | input: | (None, 20)
Dropout | output: | (None, 20)
bal_input_3 | input: | [(None, 8)]
InputLayer | output: | [(None, 8)]
concatenate_1 | input: | [(None, 20), (None, 8)]
Concatenate | output: | (None, 28)
dense_4 | input: | (None, 28)
Dense | output: | (None, 30)
dropout_4 | input: | (None, 30)
Dropout | output: | (None, 30)
dense_5 | input: | (None, 30)
Dense | output: | (None, 20)
dropout_5 | input: | (None, 20)
Dropout | output: | (None, 20)
bal_input_4 | input: | [(None, 8)]
InputLayer | output: | [(None, 8)]
concatenate_2 | input: | [(None, 20), (None, 8)]
Concatenate | output: | (None, 28)
dense_6 | input: | (None, 28)
Dense | output: | (None, 30)
dropout_6 | input: | (None, 30)
Dropout | output: | (None, 30)
dense_7 | input: | (None, 30)
Dense | output: | (None, 20)
dropout_7 | input: | (None, 20)
Dropout | output: | (None, 20)
bal_input_5 | input: | [(None, 8)]
InputLayer | output: | [(None, 8)]
concatenate_3 | input: | [(None, 20), (None, 8)]
Concatenate | output: | (None, 28)
dense_8 | input: | (None, 28)
Dense | output: | (None, 30)
dropout_8 | input: | (None, 30)
Dropout | output: | (None, 30)
dense_9 | input: | (None, 30)
Dense | output: | (None, 20)
dropout_9 | input: | (None, 20)
Dropout | output: | (None, 20)
bal_input_6 | input: | [(None, 8)]
InputLayer | output: | [(None, 8)]
concatenate_4 | input: | [(None, 20), (None, 8)]
Concatenate | output: | (None, 28)
dense_10 | input: | (None, 28)
Dense | output: | (None, 30)
input_1 | input: | [(None, 12)]
InputLayer | output: | [(None, 12)]
dropout_10 | input: | (None, 30)
Dropout | output: | (None, 30)
dense_12 | input: | (None, 12)
Dense | output: | (None, 20)
dense_11 | input: | (None, 30)
Dense | output: | (None, 20)
dropout_12 | input: | (None, 20)
Dropout | output: | (None, 20)
dropout_11 | input: | (None, 20)
Dropout | output: | (None, 20)
dense_13 | input: | (None, 20)
Dense | output: | (None, 10)
concatenate_5 | input: | [(None, 20), (None, 10)]
Concatenate | output: | (None, 30)
dense_14 | input: | (None, 30)
Dense | output: | (None, 100)
dropout_13 | input: | (None, 100)
Dropout | output: | (None, 100)
dense_15 | input: | (None, 100)
Dense | output: | (None, 50)
dropout_14 | input: | (None, 50)
Dropout | output: | (None, 50)
dense_16 | input: | (None, 50)
Dense | output: | (None, 1)

Figure 19: Sequential Model Architecture