Interim Report:
Enhancing Financial Fraud Detection through Reinforcement Learning-based Oversampling

Submitted December 5, 2023, in partial fulfillment of
the conditions for the award of the degree
BSc Computer Science with Artificial Intelligence.

Patience Chew Yee CHEAH

20314828

scypc3@nottingham.edu.cn

Supervised by Dr Boon Giin LEE

School of Computer Science University of Nottingham Ningbo China

Contents

Chapter 1

Introduction

1.1 Background and Motivation

Financial frauds, such as credit card fraud, insurance fraud and money laundering, have resulted in a considerable loss to financial institutions and the global economy [1]. Studies have designed methods for automatically detecting fraud to reduce the need for manual fraud identification. An early attempt was by adopting a rule-based expert system, which was then replaced by machine learning (ML) techniques due to the costly and ineffectiveness of the rule-based system [2]. Three recent review papers [1, 2, 3] on financial fraud discussed some popular ML methods for fraud detection, such as Support Vector Machine, Neural Networks, Random Forest and XGBoost. Despite the satisfactory achievements of these ML methods, one significant challenge raised by these papers is the imbalanced data.

Financial datasets suffer from severe class imbalance issues, where the fraudulent data are significantly lower than the legitimate ones. For example, the benchmark Kaggle Credit Card Fraud Detection dataset [4] consists of merely 0.17% fraudulent data. This skewed distribution makes the training of the ML models challenging, as the models tend to favour the majority class, causing high false-positive rates [5]. This class imbalance problem has gained more attention over the past few years, with increasing published papers on imbalanced learning topics [6].

The methods proposed to handle this problem can be categorized into data-level, algorithm-level and hybrid of these two [5, 6]. Data-level methods manipulate the dataset by removing existing majority samples (under-sampling) or generating new minority samples (oversampling). These methods are independent of classifier choice, making them flexible and easy to use. Examples of widely used oversampling methods are Synthetic Minority Oversampling Technique (SMOTE) [7], Generative Adversarial Networks (GAN) [8], and their respective variations. According to a review [3], most financial fraud detection studies applied SMOTE to increase the minority class representation. Recently, reinforcement learning (RL) has been utilized for generating or selecting training samples and has shown promising results.

Traditional oversampling techniques, which purely focus on the dataset distribution, can sometimes overfit the classifiers. Thus, it is worth designing methods that can optimize the generated samples based on the classifier's performance while keeping the choice of classifier flexible. RL, which aims to maximize cumulative reward when making sequential decisions, has the potential to optimize the generated samples. Therefore, this project will design an RL-based oversampling method to improve the classification performance of ML models.

1.2 Aims and Objectives

This project aims to tackle the class imbalance problem in the finance fraud dataset using a RL-based oversampling technique. The technique can optimize the generated samples by maximizing a reward which is defined based on the classifier's performance when trained on the augmented dataset. To achieve this aim, the objectives are:

1. To review the oversampling methods for class imbalance handling, including the utilization of RL.
2. To investigate the performances of existing ML classifiers on finance fraud detection.
3. To design an oversampling method using RL to generate high-quality samples or select suitable samples to improve the classifiers' performance.
4. To evaluate the proposed RL-based oversampling method on multiple ML classifiers and datasets.

Chapter 2

Literature Review

2.1 Oversampling Methods for Imbalanced Learning

Oversampling is a data-level method that manipulates the training dataset's distribution by generating new minority samples to balance the dataset. The standard approach is SMOTE, a technique based on random interpolation between the nearest minority samples [7]. This simple yet effective method has been applied in most finance fraud detection studies [3]. However, SMOTE may generate noisy and uninformative samples [9]. Since SMOTE purely focuses on local information, its generated samples are less diverse and realistic [10]. Thus, SMOTE's variations were proposed to tackle those challenges. Until 2018, there were more than 85 SMOTE's variations [11], including the popular Borderline-SMOTE [12], ADASYN [13] and Safe-Level SMOTE [14].

GAN [8] has been adopted for generating realistic minority samples. It has shown slightly higher recall than SMOTE on finance fraud datasets [15]. However, since GAN was originally designed for image generation, variations were proposed for oversampling tasks, such as Conditional GAN [16] and Balancing GAN [17]. Moreover, the Conditional Tabular GAN (CTGAN) [18] was proposed specifically for synthesizing tabular data such as financial data. On financial datasets, various GANs were adopted for oversampling, including IGAFN, SGAN, Wasserstein GAN and cWGAN [19]. In addition, the Unrolled GAN [20] and K-CGAN [21] outperformed SMOTE-based methods on finance fraud datasets when evaluated on F1-score. However, one challenge faced by GAN is data scarcity, as it requires a large amount of training data [10].

Recent studies have proposed hybrids of SMOTE and GAN to overcome their limitations. The SMOTified-GAN feeds the GAN's generator with SMOTE-generated minority samples instead of the commonly used random noises [10]. Another hybrid method, ESMOTE-GAN, applied SMOTE on multiple subsets of the training data, followed by GAN on the SMOTE-generated samples [9].

2.2 Reinforcement Learning Algorithms

RL is a learning paradigm in which an agent learns the best strategy that maximizes cumulative rewards based on its experience when interacting with the environment [22]. Usually, RL is formulated as a Markov Decision Process (MDP), where the agent takes an action that changes its state and produces a reward based on the quality of the new state [23]. Traditional RL algorithms like SARSA and Q-learning estimate the Q-value, that is, the best action that maximizes future rewards. The latter also takes the Q-value of the subsequent state and action into the calculation. However, these algorithms are only suitable for environments with limited states and actions [23].

Following the success of Deep Learning (DL), studies incorporate DL with RL to estimate the best action. The most popular Deep RL algorithm is the Deep Q-Network (DQN) [24] which supports unlimited states with discrete actions. However, DQN tends to select actions with the highest Q-values regardless of their optimality [23]. To address this issue, Double DQN [25] and Dueling DQN [26] were proposed. These three algorithms are value optimization-based approaches that estimate the best actions based on value function [22].

Policy optimization-based approaches, on the other hand, estimate the best action directly without using the value function. Algorithms under this category are policy gradient [27], Proximal Policy Optimization (PPO) [28] and actor-critic [22]. The actor-critic method consists of an actor that decides the best action and a critic that evaluates the action based on the new state and reward [23]. Some popular algorithms in actor-critic are Deep Deterministic Policy Gradient (DDPG) [29], Asynchronous Advantage Actor-Critic (A3C) [30] and Soft Actor-Critic (SAC) [31].

2.3 Reinforcement Learning in Imbalanced Learning

The existing works that applied RL for imbalanced learning formulated the problem in MDP. Various RL algorithms were utilized depending on the problem formulation and the suitability of the algorithms. DQN and actor-critics are used in many of these studies. These works can be categorized into data-level, algorithm-level and hybrid-level.

At the data level, RL was used to select training samples or generate minority samples. For the former, an early attempt was to optimize the training data selection using policy gradient, with the reward calculated based on the classifier's performance [32]. This method, named Trainable Undersampling (TU), outperformed traditional data-level methods like SMOTE and ADASYN. Building on TU, TRUST (TRainable Undersampling with Self Training) [33] incorporates self-training to label the unlabeled data and combines these data with the TU-selected samples to train the classifier. The study showed that TRUST can better identify minority samples compared to TU. Similar to TU, DiagSelect [34] picks the best training subset based on the classifier's performance but using the REINFORCE algorithm. For data generation, a study proposed an actor-critic method that directly generates minority samples, using a simulated reward based on the classifier's performance [35]. This method outperformed SMOTE and other distribution oversampling methods on most datasets. AutoSMOTE [36] optimizes the existing SMOTE method using hierarchical RL to decide the number of samples to generate around each minority data. Based on this decision, the agent will further decide the neighbours of the minority data and the weight to perform interpolation. AutoSMOTE significantly outperformed SMOTE-based and GAN-based oversampling methods.

At the algorithm level, DQNimb [37] is a pioneer work that utilizes a DQN as a classifier for imbalanced classification. When the agent identifies minority samples, DQNimb supplied a higher reward calculated based on the dataset's imbalanced ratio. Another study [38] used Q-learning with a similar reward function and found a reward ratio that yielded the best accuracy. In the finance fraud detection area, a study proposed the DQN method with a reward function based on the bank's revenue model and the rate of fraudulent transactions being overlooked and identified [39]. This imbalanced classification method outperformed most supervised learning models except XGBoost. Double DQN was also adopted, with a maximum positive reward given upon correct classification of fraudulent samples and a maximum negative reward upon misclassification [40]. In mechanical fault diagnosis with imbalanced data, ResDPG that applied DDPG was proposed [41]. In imbalanced multi-class classification, the DQN classifier was adopted after applying SMOTE to oversample minority class and Gaussian Mixture Model to under-sample majority class [42]. Based on these studies, the performance of RL in imbalanced classification is at least equal to the supervised learning methods.

Using only data-level and algorithm-level is insufficient to categorize all the RL-related work in imbalanced learning. AESMOTE [43] can be viewed as a hybrid of data and algorithm-level methods for imbalanced classification. In this method, two RL agents were employed, one for selecting SMOTE-generated minority samples based on the classification performance of another. The proposed hybrid method outperformed traditional undersampling methods. Another study [44] proposed an RL-based under-sampling ensemble framework that utilizes SAC to select training subsets of individual classifiers to build an ensemble classifier. This method performs better than traditional hybrid imbalanced handling techniques on datasets with severe overlapping.

Chapter 3

Methodologies

3.1 Data Preprocessing

Table 3.1 describes five benchmark finance-fraud datasets employed in this project. All the datasets are imbalanced, especially Datasets 2 and 5, shown by their percentage of minority instances. The dataset preprocessing began with removing unnecessary features (e.g., ID numbers), label encoding categorical features, and min-max scaling. The final step was to partition the datasets into training (70%), validation (10%), and testing (20%) sets without changing the original class proportions.

Table 3.1: Description of Datasets Used

No.	Dataset	Number of Features	Total Instances	Minority Percentage (%)
1.	South German Credit [45]	21	1000	30.00
2.	Africa Crisis [46]	14	1059	8.88
3.	Fake Bills [47]	7	1463	33.63
4.	Default of Credit Card Clients [48]	24	30000	22.12
5.	Credit Card Fraud Detection [4]	31	284807	0.18

3.2 Classifier Selection

Despite the superior performance of DL on image and text classification, tree-based classifiers still perform better on medium-sized tabular data [49]. Therefore, this study employed XGBoost [50], LightGBM [51], and Histogram-based Gradient Boosting Classification Tree (HGB) as classifiers. These algorithms, provided in the Python Scikit-learn library, have relatively high training speed, which eases the implementation process and training time. Due to the time constraints of this project, some experiments only used XGBoost, as it has the highest training speed.

3.3 RL Method 1: CTGAN Tuning

The GAN-based oversampling method has gained popularity in re-sampling financial fraud datasets [19]. CTGAN [18] is a GAN-based method for generating tabular data, available in a Python library. CTGAN has many hyperparameters, such as generator's and discriminator's learning rates, that can affect the quality of its generated samples. Hence, this project first attempts to tune CTGAN's hyperparameters using RL. RL has shown promising results in optimizing hyperparameters of Neural Networks and XGBoost classifiers [52, 53], but not for oversampling methods like GAN. Thus, using RL for tuning CTGAN seems to be reasonable and innovative.

Method 1 formulated the hyperparameter tuning process as a MDP. The agent perceives the current hyperparameter under consideration and decides whether to increase the hyperparameter value or proceed to the next hyperparameter. When the agent reaches the final state, it receives a reward, calculated based on the quality of the CTGAN-generated samples with the selected hyperparameter combinations. Prior research proved the independence of the order of hyperparameter selection to the RL model's performance [53]. The main components of this RL-based design are described below.

States

Table 3.2: Possible Values for Tuning CTGAN Hyperparmeters

Hyperparameters	Values
embedding_dim	4, 8, 16, 32, 64, 128
generator_dim	(4,4), (8,8), (16,16), (32,32), (64,64), (128,128)
discriminator_dim	(4,4), (8,8), (16,16), (32,32), (64,64), (128,128)
generator_decay	0.0000001, 0.0000005, 0.000001, 0.000005, 0.00001
discriminator_decay	0.0000001, 0.0000005, 0.000001, 0.000005, 0.00001
generator_lr	0.000001, 0.00005, 0.0001, 0.0005, 0.001, 0.005
discriminator_lr	0.000001, 0.00005, 0.0001, 0.0005, 0.001, 0.005

A state represents a CTGAN hyperparameter and its selected value, in which the possible values are listed in Table 3.2. The final state indicates that the agent has completed all hyperparameter selections. The agent may deadlock in a particular hyperparameter (i.e., keeps increasing a hyperparameter already at its maximum). Hence, when the agent reaches a step limit, it will automatically transit to the final step.

Actions

Two discrete actions are employed to simplify the design: Increase the current hyperparameter's value or proceed to the next hyperparameter. The hyperparameter selection always begins with the smallest value, and the agent keeps increasing it until it reaches the desired value. Even without decrement, the agent can still reach all possible hyperparameter combinations. Thus, only two actions are allowed.

Agent

The Keras-RL library was used to implement RL agents. Only three RL agents in this library support discrete action space: DQN [24] (also supports Double and Dueling networks [25, 26]), CEM [54], SARSA [55]. Thus, these agents were employed, with their hyperparameters shown in Table B.1. All three agents are constructed with two fully connected hidden layers, with ReLU activation and 16 neurons each. The output layer employs linear activation. Preliminary experiments show that episode rewards do not change much during the training. Thus, the agents only trained for 100 training episodes.

Reward

A reward function must guide the agent to improve the classifier's performance by enhancing the quality of CTGAN-generated minority samples. Two crucial performance metrics for finance fraud detection are the recall (on minority class) and AUC, as the goal is to detect as many frauds as possible while maintaining a low false-positive. Since the goal is to improve both recall and AUC, the final reward (R) is calculated as:

$$R = ((AUC_f - AUC_i) + (RC_f - RC_i)) \times 10^3 \tag{3.1}$$

where AUC_i and RC_i represent the classifier's AUC and recall, respectively, when trained with minority samples generated using default CTGAN hyperparameters. AUC_f and RC_f denote AUC and recall achieved when employing the selected CTGAN hyperparameters. A scalar factor of 10^3 is applied to amplify the reward value.

Providing rewards only at the final state may cause sparse or delayed rewards, hindering the agent's learning process. Hence, small rewards are issued in each intermediate step based on the average quality of the currently selected hyperparameter value. Specifically, a penalty of -5 is applied if the agent takes an invalid action (i.e., increasing a hyperparameter already at its maximum value). If there is an insufficient historical record, there is no way to deduce the effect of the selected hyperparameter value on the final performance. Thus, no reward is provided. If there is sufficient historical information, a reward of 1 will be supplied if there are improvements in both recall and AUC. Otherwise, a penalty of -1 will be given. The whole reward calculation algorithm is detailed in Algorithm 1.

3.4 RL Method 2: Sample Selection

The performance of RL Method 1 is below expectation. One possible reason is that CTGAN with hyperparameter tuning may still generate noisy samples. Therefore, this project continues with the sample selection method to remove the noises.

Before applying RL to the design, an Euclidean-based sample selection method was tested. This simple method computes the Euclidean distance between the generated samples with both majority and minority original data. It discards all the generated samples that are either too far away from the minority clusters or too close to the majority. The actual implementation is provided in Algorithm A. This method can slightly improve the recall rates, as shown in Table C.3. However, the enhancements are insignificant and require manual tuning. Moreover, apart from the distance between the generated samples and the original data, other factors may influence the classification performance.

Thus, an RL-based method was designed, aiming to automatically select the 'high quality' samples that lead to the optimal classification performance. The agent will decide to keep or discard the generated samples and receive feedback based on the classifier's performance when trained on the selected samples. The method is designed in a general way that can suit all oversampling approaches. Nevertheless, the experiments were only carried out using CTGAN and SMOTE.

States

A state represents a generated sample under consideration and a value that indicates the current amount of selected samples. All the values were normalized in the range [-1, 1].

Actions

For each CTGAN-generated sample, the agent can take one of the actions: Keep or discard the generated training sample.

Agent

Since Keras-RL library does not provide advanced RL algorithms, such as actor-critic and PPO, Stable-Baseline3 was employed. Three RL agents that support discrete actions were adopted: DQN, A2C and PPO. Table B.2 detailed their hyperparameters.

Reward

Unlike Method 1, the reward is issued only when all the samples have been decided. Even though the sparse and delayed reward may cause difficulties in training the agent, no better intermediate reward function has come out yet.

To further simplify the agent's goal, only recall is considered when designing the reward function. The agent will aim to enhance the recall of the classifier. Algorithm 2 demonstrates the reward calculation, where the final reward is normalized within -1 to 1. It is clear from the algorithm that the agent will gain a positive reward if the recall improves, otherwise, the reward is negative.

Chapter 4

Preliminary Results

The classification performance with and without oversampling methods, evaluated on five selected datasets, are recorded in Table C.1. SMOTE and CTGAN (with default hyperparameters provided by the Python library) were used for oversampling. The evaluation metrics used are Precision (PR), Recall (RC), F1-score (F1) and Area Under Curve (AUC).

4.1 Results For Method 1

The performance after hyperparameter tuning using Method 1, with three different RL agents, is recorded in Table C.2. Table 4.1 demonstrated the performance improvements after CTGAN hyperparameter tuning, calculated with respect to the performance achieved before tuning. The RC did not improve substantially after hyperparameter tuning. The enhancement of AUC is close to negligible. Surprisingly, the PR and F1 exhibit relatively higher improvements, although the reward function does not consider these metrics. The overall improvements across all metrics are insignificant. This result does not align with the initial expectation that the RL agent would identify optimal CTGAN hyperparameter configurations.

Table 4.1: Analysis on Improvements After CTGAN Hyperparameter Tuning

Improvement	PR	RC	F1	AUC
Sum	0.52	0.28	0.40	0.03
Maximum	0.25	0.08	0.13	0.04
Minimum	-0.09	-0.07	-0.05	-0.03
Mean	0.01	0.01	0.01	0.00

Several factors may account for this sub-optimal result. The inefficient reward function and hyperparameter settings of the RL agent may cause agents to fail to learn. The different distributions of validation and testing datasets may cause the agent to overfit the validation set. Furthermore, the CTGAN hyperparameters may not significantly affect the quality of its generated samples. Under the same hyperparameters, CTGAN may generate different samples and possibly contain noises.

4.2 Results For Method 2

Table C.3 demonstrates the performance of Euclidean-based CTGAN sample selection method (Algorithm A), when evaluated using XGBoost on the South German Credit dataset. At certain hyperparameter combinations, the method can achieve slightly higher RC than the original CTGAN without sample selection. However, this method requires manual hyperparameter selection to achieve satisfactory performance, and the improvement is insignificant. Nevertheless, the results provide an insight that, on the South German Credit dataset, it is possible to manipulate the dataset distribution to achieve 0.62% RC and 0.76% AUC, using the XGBoost classifier.

RL agents (DQN, A2C, PPO) were trained to select SMOTE and CTGAN-generated samples. Their training mean episode reward is displayed in Figure B.1. Over 260k timesteps, the episode reward fluctuates but there is no significant increment. Hence, it is unsure if the agents have learnt properly.

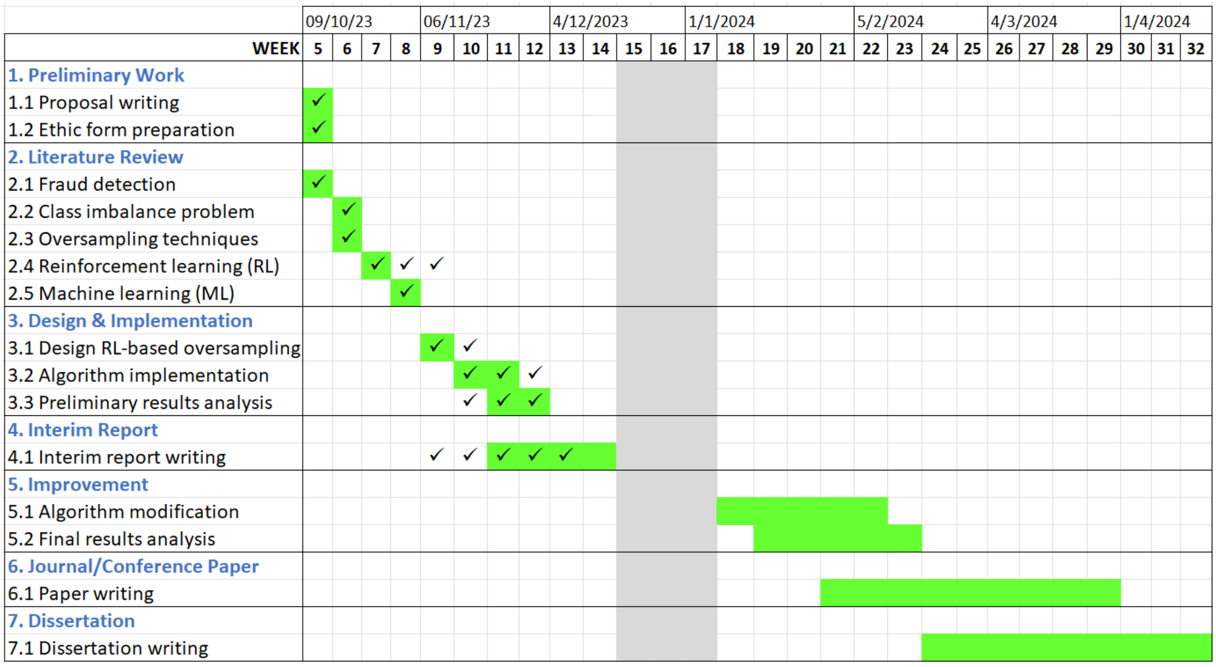
The agents at every 100 episodes (28k timesteps) were saved and evaluated. Table C.4 and C.5 show the number of samples selected on each 100-elapsed episode and their corresponding classification performance. Among nine different time stamps, the best-trained models were chosen based on the RC. The best models and the performance are summarized in Table 4.2. Based on the summary, there are minor improvements across all four metrics, but the improvements are insignificant. Although the reward function was calculated based on RC, this metric did not improve much. Similar to Method 1, the agents may not learn effectively due to the poor design of the reward function and the observation space. The difference between validation and testing dataset distributions may lead to the overfitting issue.

Table 4.2: Summary of RL Method 2 on South German Credit Dataset With XGBoost

Oversampling	Agent	PR	RC	F1	AUC
SMOTE	-	0.51	0.58	0.54	0.75
	DQN	0.54	0.62	0.57	0.77
	A2C	0.54	0.55	0.55	0.76
	PPO	0.55	0.60	0.57	0.75
CTGAN	-	0.49	0.57	0.53	0.73
	DQN	0.52	0.58	0.55	0.74
	A2C	0.52	0.57	0.54	0.75
	PPO	0.49	0.58	0.53	0.74

Chapter 5

Progress



	09/10/23				06/11/23				4/12/2023				1/1/2024					5/2/2024				4/3/2024				1/4/2024		
WEEK	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**
1. Preliminary Work											▨	▨	▨															
1.1 Proposal writing	🟩✓										▨	▨	▨															
1.2 Ethic form preparation	🟩✓										▨	▨	▨															
2. Literature Review											▨	▨	▨															
2.1 Fraud detection	🟩✓										▨	▨	▨															
2.2 Class imbalance problem		🟩✓									▨	▨	▨															
2.3 Oversampling techniques		🟩✓									▨	▨	▨															
2.4 Reinforcement learning (RL)			🟩✓	✓	✓						▨	▨	▨															
2.5 Machine learning (ML)				🟩✓							▨	▨	▨															
3. Design & Implementation											▨	▨	▨															
3.1 Design RL-based oversampling					🟩✓	✓					▨	▨	▨															
3.2 Algorithm implementation						🟩✓	🟩✓	✓			▨	▨	▨															
3.3 Preliminary results analysis						✓	🟩✓	🟩✓			▨	▨	▨															
4. Interim Report											▨	▨	▨															
4.1 Interim report writing					✓	✓	🟩✓	🟩✓	🟩✓	🟩	▨	▨	▨															
5. Improvement											▨	▨	▨															
5.1 Algorithm modification											▨	▨	▨	🟩	🟩	🟩	🟩	🟩										
5.2 Final results analysis											▨	▨	▨		🟩	🟩	🟩	🟩	🟩									
6. Journal/Conference Paper											▨	▨	▨															
6.1 Paper writing											▨	▨	▨				🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩			
7. Dissertation											▨	▨	▨															
7.1 Dissertation writing											▨	▨	▨							🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩

Figure 5.1: Updated Project Time Plan and Completion of Milestones

Figure 5.1 shows the updated project plan with completed milestones marked with check marks. Overall, the project was well managed, and the tasks were completed on schedule. The literature review on the relevant fields is summarized in Sections 1 and 2. Two designs of RL-based oversampling methods, hyperparameter tuning and sample selection, are described in Section 3. Their results are discussed in Section 4. The hyperparameter tuning method and its preliminary results were submitted to the 2nd IEEE TEMSCON ASPAC (Technology and Engineering Management Society Conference: Asia-Pacific) and was accepted for presentation.

The remaining project will focus on modifying the designed algorithm to enhance its final performance. It is planned to submit the results to a conference or a journal. All remaining tasks will be started one week earlier than originally planned to ensure there is sufficient time to complete the tasks with better quality.

Chapter 6

Reflection

Since the summer vacation, preliminary work has been started, including background study, initial design and implementation. The prior knowledge and experience have made the project progress smoothly, yet some challenges persist. Implementing the designed RL framework is challenging. State-of-the-art works in the related area have shown promising results in imbalanced learning using RL. However, since most research papers do not provide source codes, it was unable to re-implement their works. Hence, this project only implements some basic and simple methods. Even though there are several Python libraries for RL, choosing the most suitable one is not straightforward. Initially, Keras-RL was utilized as it is flexible in defining the agent's network and can train the agent in a single line of code. However, more advanced RL algorithms, such as actor-critic and PPO, are unavailable. Thus, Stable-Baseline3 was used in the middle of this project, as this library provides advanced algorithms. The switching of the library choice has raised compatibility issues with the previous implementation. Fortunately, this issue has been resolved after modifying the initial codes.

The greatest challenge is the design of the RL-based method. The results from the implementation were always under-satisfactory, which may be due to the improper design of the reward function or observation space. The agent may be unable to learn from this environment design. Despite multiple attempts to redesign the reward function and modify the observation space, the performance still did not improve. Furthermore, the hyperparameter choice of RL can also affect its training performance. However, it is challenging to determine the best hyperparameters manually or automatically. Hence, future work will focus on improving the agent's performance by modifying the current design and the hyperparameters.

References

[1] W. Hilal, S. A. Gadsden, and J. Yawney, "Financial fraud: A review of anomaly detection techniques and recent advances," *Expert Systems with Applications*, vol. 193, p. 116429, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417421017164

[2] X. Zhu, X. Ao, Z. Qin, Y. Chang, Y. Liu, Q. He, and J. Li, "Intelligent financial fraud detection practices in post-pandemic era," *The Innovation*, vol. 2, no. 4, p. 100176, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666675821001016

[3] A. Ali, S. Abd Razak, S. H. Othman, T. A. E. Eisa, A. Al-Dhaqm, M. Nasser, T. Elhassan, H. Elshafie, and A. Saif, "Financial fraud detection based on machine learning: A systematic literature review," *Applied Sciences*, vol. 12, no. 19, p. 9637, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/19/9637

[4] "Credit card fraud detection," 2018. [Online]. Available: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

[5] B. Krawczyk, "Learning from imbalanced data: open challenges and future directions," *Progress in Artificial Intelligence*, vol. 5, pp. 221–232, 2016.

[6] S. Rezvani and X. Wang, "A broad review on class imbalance learning techniques," *Applied Soft Computing*, vol. 143, p. 110415, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494623004337

[7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Int. Res.*, vol. 16, no. 1, p. 321–357, 2002.

[8] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, p. 2672–2680.

[9] F. A. Ghaleb, F. Saeed, M. Al-Sarem, S. N. Qasem, and T. Al-Hadhrami, "Ensemble synthesized minority oversampling-based generative adversarial networks and random forest algorithm for credit card fraud detection," *IEEE Access*, vol. 11, pp. 89 694–89 710, 2023.

[10] A. Sharma, P. K. Singh, and R. Chandra, "SMOTified-GAN for class imbalanced pattern classification problems," *IEEE Access*, vol. 10, pp. 30 655–30 665, 2022.

[11] A. Fernández, S. García, F. Herrera, and N. V. Chawla, “SMOTE for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary,” *J. Artif. Int. Res.*, vol. 61, no. 1, p. 863–905, jan 2018.

[12] H. Han, W.-Y. Wang, and B.-H. Mao, “Borderline-smote: A new over-sampling method in imbalanced data sets learning,” in *Advances in Intelligent Computing*, D.-S. Huang, X.-P. Zhang, and G.-B. Huang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 878–887.

[13] H. He, Y. Bai, E. A. Garcia, and S. Li, “Adasyn: Adaptive synthetic sampling approach for imbalanced learning,” in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, 2008, pp. 1322–1328.

[14] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap, “Safe-Level-SMOTE: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem,” in *Advances in Knowledge Discovery and Data Mining*, T. Theeramunkong, B. Kijsirikul, N. Cercone, and T.-B. Ho, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 475–482.

[15] U. Fiore, A. De Santis, F. Perla, P. Zanetti, and F. Palmieri, “Using generative adversarial networks for improving classification effectiveness in credit card fraud detection,” *Information Sciences*, vol. 479, pp. 448–455, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025517311519

[16] G. Douzas and F. Bacao, “Effective data generation for imbalanced learning using conditional generative adversarial networks,” *Expert Systems with Applications*, vol. 91, pp. 464–471, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417417306346

[17] G. Mariani, F. Scheidegger, R. Istrate, C. Bekas, and C. Malossi, “Bagan: Data augmentation with balancing gan,” 2018.

[18] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, “Modeling tabular data using conditional GAN,” in *Advances in Neural Information Processing Systems*, 2019.

[19] R. Sauber-Cole and T. M. Khoshgoftaar, “The use of generative adversarial networks to alleviate class imbalance in tabular data: a survey,” *J. Big Data*, vol. 9, no. 1, Aug. 2022.

[20] J. Wang and L. Yao, “Unrolled gan-based oversampling of credit card dataset for fraud detection,” in *2022 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, 2022, pp. 858–861.

[21] E. Strelcenia and S. Prakoonwit, “A new gan-based data augmentation method for handling class imbalance in credit card fraud detection,” in *2023 10th International Conference on Signal Processing and Integrated Networks (SPIN)*, 2023, pp. 627–634.

[22] A. K. Shakya, G. Pillai, and S. Chakrabarty, “Reinforcement learning algorithms: A brief survey,” *Expert Systems with Applications*, vol. 231, p. 120495, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417423009971

[23] F. AlMahamid and K. Grolinger, “Reinforcement learning algorithms: An overview and classification,” in *2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2021, pp. 1–7.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” 2013.

[25] D. S. Hado van Hasselt, Arthur Guez, “Deep reinforcement learning with double Q-learning,” vol. 30, 2016.

[26] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, “Dueling network architectures for deep reinforcement learning,” 2016.

[27] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, ser. NIPS’99. Cambridge, MA, USA: MIT Press, 1999, p. 1057–1063.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” 2017.

[29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” 2019.

[30] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML’16. JMLR.org, 2016, p. 1928–1937.

[31] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” 2018.

[32] M. Peng, Q. Zhang, X. Xing, T. Gui, X. Huang, Y.-G. Jiang, K. Ding, and Z. Chen, “Trainable undersampling for class-imbalance learning,” *Proc. Conf. AAAI Artif. Intell.*, vol. 33, no. 01, pp. 4707–4714, Jul. 2019.

[33] J. Chi, G. Zeng, Q. Zhong, T. Liang, J. Feng, X. Ao, and J. Tang, “Learning to undersampling for class imbalanced credit risk forecasting,” in *2020 IEEE International Conference on Data Mining (ICDM)*, 2020, pp. 72–81.

[34] S. Fan, X. Zhang, and Z. Song, “Imbalanced sample selection with deep reinforcement learning for fault diagnosis,” *IEEE Transactions on Industrial Informatics*, vol. 18, no. 4, pp. 2518–2527, 2022.

[35] Y. Zhou, J. Shu, X. Zhong, X. Huang, C. Luo, and J. Ai, “Oversampling algorithm based on reinforcement learning in imbalanced problems,” in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 01–06.

[36] D. Zha, K.-H. Lai, Q. Tan, S. Ding, N. Zou, and X. Hu, “Towards automated imbalanced learning with deep hierarchical reinforcement learning,” in *CIKM*, 2022.

[37] E. Lin, Q. Chen, and X. Qi, “Deep reinforcement learning for imbalanced classification,” *Applied Intelligence*, vol. 50, no. 8, p. 2488–2502, 2020. [Online]. Available: https://doi.org/10.1007/s10489-020-01637-z

[38] L. Zhinin-Vera, O. Chang, R. Valencia-Ramos, R. Velastegui, G. Pilliza, and F. Socasi, "Q-credit card fraud detector for imbalanced classification using reinforcement learning," 02 2020, pp. 279–286.

[39] S. Vimal, K. Kayathwal, H. Wadhwa, and G. Dhama, "Application of deep reinforcement learning to payment fraud," 2021.

[40] M. Jacob, G. S. H. Reddy, C. Rappai, P. Kapoor, and M. Kolhekar, "A comparative study of supervised and reinforcement learning techniques for the application of credit defaulters," in *2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT)*, 2022, pp. 1–6.

[41] Q. Cui, L. Zhu, H. Feng, S. He, and J. Chen, "Intelligent fault quantitative identification via the improved deep deterministic policy gradient (ddpg) algorithm accompanied with imbalanced sample," *IEEE Transactions on Instrumentation and Measurement*, vol. 72, pp. 1–13, 2023.

[42] Y.-J. Su, L.-C. Wu, C.-H. Chen, and T.-Y. Chen, "Combining data resampling and drl algorithm for intrusion detection," in *2023 5th International Conference on Computer Communication and the Internet (ICCCI)*, 2023, pp. 47–51.

[43] X. Ma and W. Shi, "AESMOTE: Adversarial reinforcement learning with SMOTE for anomaly detection," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 943–956, 2021.

[44] M. Qian and Y.-F. Li, "A novel adaptive undersampling framework for class-imbalance fault detection," *IEEE Transactions on Reliability*, vol. 72, no. 3, pp. 1003–1017, 2023.

[45] "South german credit," 2019. [Online]. Available: https://doi.org/10.24432/C5X89F

[46] C. Reinhart, K. Rogoff, C. Trebesch, and V. Reinhart, "Africa economic, banking and systemic crisis data," 2019. [Online]. Available: https://www.kaggle.com/datasets/chirin/africa-economic-banking-and-systemic-crisis-data

[47] "Fake bills," 2019. [Online]. Available: https://www.kaggle.com/datasets/alexandrepetit881234/fake-bills

[48] I.-C. Yeh, "Default of credit card clients," 2016. [Online]. Available: https://doi.org/10.24432/C55S3H

[49] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 507–520. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/0378c7692da36807bdec87ab043cdadc-Paper-Datasets_and_Benchmarks.pdf

[50] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 785–794. [Online]. Available: https://doi.org/10.1145/2939672.2939785

[51] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 3149–3157.

[52] H. S. Jomaa, J. Grabocka, and L. Schmidt-Thieme, "Hyp-rl : Hyperparameter optimization by reinforcement learning," 2019.

[53] J. Wu, S. Chen, and X. Liu, "Efficient hyperparameter optimization through model-based reinforcement learning," *Neurocomputing*, vol. 409, pp. 381–393, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231220310523

[54] I. Szita and A. Lörincz, "Learning tetris using the noisy cross-entropy method," *Neural Computation*, vol. 18, no. 12, pp. 2936–2941, 2006.

[55] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

Appendix A

Important Algorithms

Algorithm 1 Reward Calculation For RL Method 1

```
if agent in final state then
    $R \leftarrow ((AUC_f - AUC_i) + (RC_f - RC_i)) \times 10^3$
else
    if action is invalid then                    ▷ Trying to increase a maximum value
        $R \leftarrow -5$
    else if No. of historical record with same hyperparameter value $\leq 3$ then
        $R \leftarrow 0$
    else
        $I \leftarrow (Avg(RC_f - RC_i) + Avg(AUC_f - AUC_i)) \times 100$
        if $I > 0$ then
            $R \leftarrow 1$
        else
            $R \leftarrow -1$
        end if
    end if
end if
```

Algorithm 2 Reward Calculation For RL Method 2

```
if agent in final state then
    $RC_{imp} \leftarrow RC_f - RC_i$
    if $RC_{imp} > 0$ then
        $R \leftarrow \frac{RC_{imp}}{1-RC_i}$
    else
        $R \leftarrow \frac{RC_{imp}}{RC_i}$
    end if
else
    $R \leftarrow 0$
end if
```

Algorithm 3: Euclidean-Based Sample Selection

(Python code snippet using list comprehension)

```python
"""
    Input:
    - ori_min: Original minority data
    - ori_maj: Original majority data
    - gen: Generated minority samples
    - k_min: No. of nearest original minority data to be considered,
            with respect to a particular generated sample
    - k_maj: No. of nearest original majority data to be considered,
            with respect to a particular generated sample
    - dist_min: Maximum distance between generated sample and original
            minority data
    - dist_maj: Minimum distance between generated sample and original
            majority data

    Algorithm:
        Filters out all the generated samples that are too far away
        from original minority data, or too near to majority data

    Output: Selected generated samples (i.e. a subset of gen)
"""

def euclidean_based_select2(ori_min, ori_maj, gen, k_min=5, k_maj=5,
   dist_min=1.5, dist_maj=1.5):

    len_min = len(ori_min)
    len_maj = len(ori_maj)
    len_gen = len(gen)

    sel = [
        np.average(np.sort([math.dist(gen[i], ori_min[j])
        for j in range(len_min)])[:k_min]) < dist_min
        and np.average(np.sort([math.dist(gen[i], ori_maj[m])
        for m in range(len_maj)])[:k_maj]) > dist_maj
        for i in range(len_gen)
    ]

    return gen[np.array(sel) == True]
```

Appendix B

RL Agents

Table B.1: Hyperparameters of RL Agents (RL Method 1)

Agent	Hyperparameters
DQN (Double Dueling)	Sequential memory with limit 700 Policy: BoltzmannQ Warm-up steps: 300 Target model update: 1 Gamma: 1.0 Optimizer: Adam (learning rate = 0.005) Metrics: MAE
CEM	Episode Parameter Memory with limit 700 Warm-up steps: 1000
SARSA	Policy: BoltzmannQ Warm-up steps: 300 Gamma: 1.0 Optimizer: Adam

Table B.2: Hyperparameters of RL Agents (RL Method 2)

Agent	Hyperparameters
DQN	MlpPolicy, learning_rate=0.0001, gamma=0.99
A2C	MlpPolicy, learning_rate=0.0007, gamma=0.99
PPO	MlpPolicy, learning_rate=0.0003, gamma=0.99, n_epochs=10

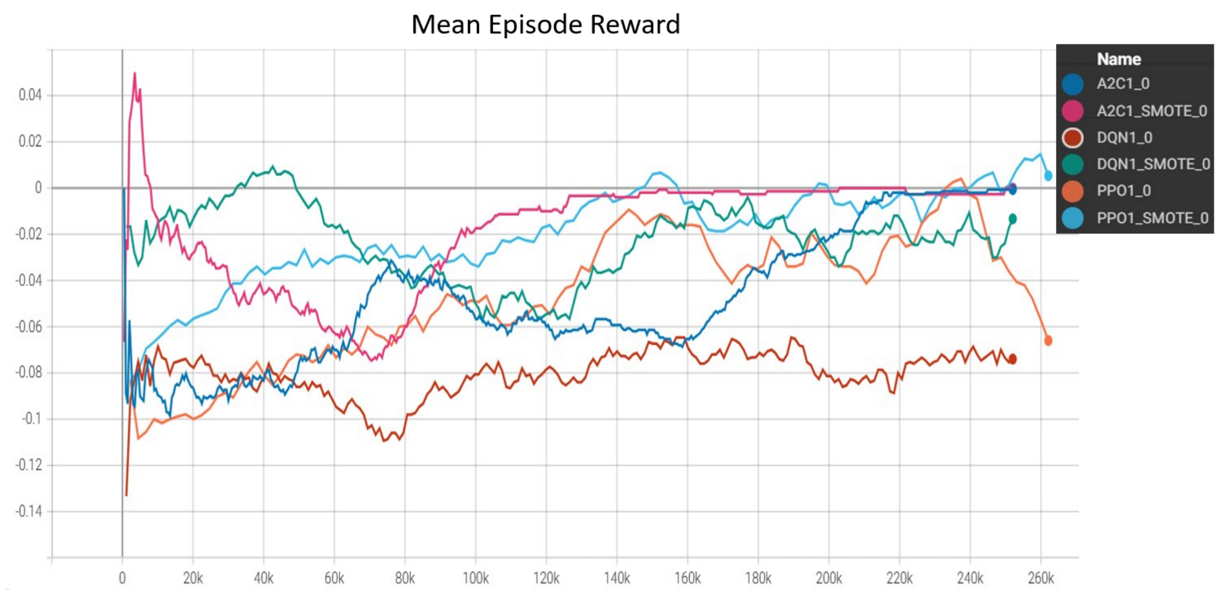


Figure B.1: Mean Episode Reward When Training Agents on RL Method 2

Appendix C

Results

Table C.1: Classifiers' Performance On Different Generated Minority Samples

Dataset	Oversampling	Classifier	PR	RC	F1	AUC
South German Credit	Original	XGBoost	0.54	0.55	0.55	0.76
		LightGBM	0.55	0.57	0.56	0.75
		HGB	0.50	0.53	0.52	0.74
	SMOTE	XGBoost	0.51	0.58	0.54	0.75
		LightGBM	0.52	0.52	0.52	0.73
		HGB	0.50	0.53	0.52	0.74
	CTGAN	XGBoost	0.49	0.57	0.53	0.73
		LightGBM	0.51	0.50	0.50	0.74
		HGB	0.55	0.58	0.56	0.76
African Crisis	Original	XGBoost	0.95	0.95	0.95	1.00
		LightGBM	0.89	0.89	0.89	1.00
		HGB	0.94	0.89	0.92	1.00
	SMOTE	XGBoost	0.83	1.00	0.90	1.00
		LightGBM	0.86	1.00	0.93	1.00
		HGB	0.83	1.00	0.90	1.00
	CTGAN	XGBoost	0.95	1.00	0.97	1.00
		LightGBM	0.90	0.95	0.92	1.00
		HGB	0.95	1.00	0.97	1.00
Fake Bills	Original	XGBoost	0.99	0.95	0.97	1.00
		LightGBM	1.00	0.94	0.97	1.00
		HGB	0.99	0.93	0.96	1.00
	SMOTE	XGBoost	0.99	0.95	0.97	1.00
		LightGBM	1.00	0.93	0.96	1.00
		HGB	0.99	0.93	0.96	1.00
	CTGAN	XGBoost	0.99	0.96	0.97	0.99
		LightGBM	0.97	0.95	0.96	0.99
		HGB	0.99	0.93	0.96	1.00
Default of Credit Card Clients	Original	XGBoost	0.63	0.37	0.46	0.76
		LightGBM	0.68	0.37	0.48	0.78
		HGB	0.68	0.37	0.48	0.78
	SMOTE	XGBoost	0.56	0.42	0.48	0.74
		LightGBM	0.60	0.43	0.50	0.76
		HGB	0.58	0.46	0.51	0.76
	CTGAN	XGBoost	0.63	0.37	0.46	0.76
		LightGBM	0.67	0.36	0.47	0.78
		HGB	0.66	0.38	0.48	0.78
Credit Card Fraud	Original	XGBoost	0.92	0.71	0.80	0.97
		LightGBM	0.34	0.63	0.44	0.82
		HGB	0.51	0.24	0.33	0.39
	SMOTE	XGBoost	0.79	0.81	0.80	0.98
		LightGBM	0.54	0.83	0.66	0.99
		HGB	0.51	0.84	0.63	0.98
	CTGAN	XGBoost	0.95	0.74	0.83	0.97
		LightGBM	0.65	0.72	0.69	0.98
		HGB	0.79	0.71	0.75	0.98

Table C.2: Performance of Classifiers After CTGAN Hyperparameter Tuning

Dataset	Classifier	Agent	PR	RC	F1	AUC
South German Credit	HGB	DQN	0.52	0.52	0.52	0.73
		CEM	0.52	0.53	0.52	0.74
		SARSA	0.53	0.53	0.53	0.74
	LightGBM	DQN	0.52	0.58	0.55	0.75
		CEM	0.47	0.55	0.51	0.73
		SARSA	0.50	0.53	0.52	0.75
	XGBoost	DQN	0.54	0.60	0.57	0.76
		CEM	0.55	0.57	0.56	0.76
		SARSA	0.49	0.53	0.51	0.72
Africa Crisis	HGB	DQN	0.86	1.00	0.93	1.00
		CEM	0.90	1.00	0.95	1.00
		SARSA	0.90	0.95	0.92	1.00
	LightGBM	DQN	0.90	0.95	0.92	1.00
		CEM	0.95	1.00	0.97	1.00
		SARSA	0.95	1.00	0.97	1.00
	XGBoost	DQN	0.95	1.00	0.97	1.00
		CEM	0.90	1.00	0.95	1.00
		SARSA	0.95	1.00	0.97	1.00
Fake Bills	HGB	DQN	0.99	0.95	0.97	1.00
		CEM	0.99	0.92	0.95	1.00
		SARSA	0.96	0.96	0.96	1.00
	LightGBM	DQN	1.00	0.93	0.96	1.00
		CEM	1.00	0.93	0.96	1.00
		SARSA	0.97	0.96	0.96	0.99
	XGBoost	DQN	0.96	0.97	0.96	1.00
		CEM	0.99	0.95	0.97	1.00
		SARSA	0.99	0.95	0.97	1.00
Default of Credit Card Clients	HGB	DQN	0.68	0.37	0.48	0.78
		CEM	0.66	0.37	0.48	0.78
		SARSA	0.67	0.37	0.48	0.78
	LightGBM	DQN	0.68	0.37	0.48	0.78
		CEM	0.67	0.37	0.48	0.78
		SARSA	0.67	0.37	0.48	0.77
	XGBoost	DQN	0.63	0.36	0.46	0.76
		CEM	0.63	0.37	0.46	0.76
		SARSA	0.62	0.38	0.47	0.77
Credit Card Fraud Detection	HGB	DQN	0.93	0.71	0.81	0.98
		CEM	0.73	0.79	0.76	0.97
		SARSA	0.85	0.74	0.79	0.97
	LightGBM	DQN	0.90	0.73	0.81	0.97
		CEM	0.88	0.77	0.82	0.98
		SARSA	0.88	0.71	0.79	0.97
	XGBoost	DQN	0.87	0.78	0.82	0.98
		CEM	0.86	0.79	0.82	0.98
		SARSA	0.91	0.73	0.81	0.98

Table C.3: XGBoost's Performance With Algorithm 3 on South German Credit Dataset, Using Different Hyperparameters.

k_min	k_maj	dist_min	dist_maj	No. Samples Selected	PR	RC	F1	AUC
10	15	0.9	0.8	22	0.54	0.60	0.57	0.76
10	10	1.4	0.7	280	0.51	0.62	0.56	0.76
10	10	1.1	0.8	180	0.51	0.58	0.55	0.76
10	15	1.4	0.7	280	0.51	0.62	0.56	0.76
10	15	1	0.9	70	0.52	0.60	0.56	0.76
10	15	1.3	0.9	280	0.53	0.60	0.56	0.75
10	15	1.4	1	280	0.53	0.62	0.57	0.75
10	20	1.4	0.7	280	0.54	0.62	0.58	0.75
10	20	1.4	0.8	280	0.51	0.62	0.56	0.76
10	20	1	0.9	74	0.55	0.58	0.56	0.75
10	20	1.2	1	280	0.54	0.58	0.56	0.75
10	20	1.4	1	280	0.52	0.58	0.55	0.75
15	10	1.4	0.8	280	0.53	0.58	0.56	0.75
15	15	1.4	0.8	280	0.53	0.58	0.56	0.75
15	15	1.4	0.9	280	0.53	0.62	0.57	0.75
15	15	1.1	1	81	0.55	0.60	0.57	0.75
15	20	1.3	0.9	280	0.54	0.58	0.56	0.75
20	10	1.1	0.7	95	0.53	0.62	0.57	0.75
20	15	1.1	0.7	95	0.53	0.62	0.57	0.75
20	15	1.1	0.8	94	0.55	0.60	0.58	0.75
20	20	1.2	0.7	195	0.52	0.60	0.56	0.76
20	20	1.1	0.8	95	0.53	0.62	0.57	0.75
20	20	1.3	0.9	280	0.56	0.60	0.58	0.75
20	20	1.4	1	280	0.52	0.60	0.56	0.75

Algorithm 3 takes 1000 CTGAN generated samples as input, and selects samples that satisfy the requirements (determined by the hyperparameter configurations). Only the first 280 samples were used for training. The whole results, with over 680 hyperparameter combinations, cannot fit into a single table. Thus, this table is filtered by $RC > 0.57, AUC > 0.74, PR > 0.5$.

Table C.4: XGBoost's Performance With RL Method 2 on South German Credit Dataset Tested on Multiple Training Episodes. Agent Selects From 280 SMOTE-Generated Samples.

Agent	Episode	No. Samples Selected	PR	RC	F1	AUC
DQN	100	37	0.55	0.60	0.57	0.78
	200	119	0.54	0.62	0.57	0.77
	300	174	0.50	0.58	0.54	0.72
	400	160	0.51	0.57	0.54	0.74
	500	92	0.50	0.55	0.52	0.74
	600	134	0.48	0.62	0.54	0.74
	700	52	0.48	0.60	0.53	0.75
	800	186	0.48	0.53	0.50	0.72
	900	119	0.49	0.57	0.52	0.72
A2C	100	0	0.54	0.55	0.55	0.76
	200	62	0.44	0.52	0.47	0.71
	300	0	0.54	0.55	0.55	0.76
	400	0	0.54	0.55	0.55	0.76
	500	0	0.54	0.55	0.55	0.76
	600	0	0.54	0.55	0.55	0.76
	700	0	0.54	0.55	0.55	0.76
	800	0	0.54	0.55	0.55	0.76
	900	0	0.54	0.55	0.55	0.76
PPO	100	107	0.53	0.60	0.56	0.75
	200	91	0.47	0.55	0.51	0.75
	300	76	0.50	0.55	0.52	0.75
	400	67	0.55	0.60	0.57	0.75
	500	51	0.50	0.58	0.54	0.75
	600	44	0.49	0.57	0.52	0.75
	700	48	0.57	0.58	0.58	0.77
	800	50	0.52	0.60	0.56	0.75
	900	45	0.52	0.55	0.53	0.75

Table C.5: XGBoost's Performance With RL Method 2 on South German Credit Dataset Tested on Multiple Training Episodes. Agent Selects From 280 CTGAN-Generated Samples.

Agent	Episode	No. Samples Selected	PR	RC	F1	AUC
DQN	100	0	0.53	0.55	0.54	0.75
	200	133	0.48	0.54	0.51	0.74
	300	95	0.52	0.58	0.55	0.74
	400	75	0.49	0.56	0.52	0.74
	500	69	0.49	0.56	0.52	0.73
	600	86	0.48	0.54	0.51	0.73
	700	66	0.50	0.56	0.53	0.74
	800	86	0.52	0.56	0.54	0.75
	900	80	0.51	0.55	0.53	0.74
A2C	100	163	0.50	0.57	0.53	0.75
	200	57	0.49	0.54	0.51	0.74
	300	20	0.50	0.56	0.53	0.75
	400	12	0.50	0.55	0.52	0.74
	500	8	0.50	0.55	0.53	0.75
	600	1	0.52	0.57	0.54	0.75
	700	0	0.54	0.55	0.55	0.75
	800	0	0.54	0.55	0.55	0.76
	900	0	0.54	0.55	0.55	0.75
PPO	100	0	0.54	0.55	0.55	0.76
	200	4	0.51	0.56	0.53	0.75
	300	5	0.50	0.56	0.53	0.75
	400	8	0.49	0.52	0.51	0.75
	500	9	0.49	0.55	0.52	0.74
	600	5	0.50	0.55	0.52	0.75
	700	8	0.51	0.56	0.53	0.75
	800	17	0.49	0.58	0.53	0.74
	900	5	0.50	0.57	0.53	0.75