Universiteit Leiden

Master Computer Science

Modern Machine Learning for Anti-Money Laundering

Name:	Jingsen Chen
Student ID:	s2434792
Date:	21/04/2021
Specialisation:	Advanced Data Analytics
1st supervisor:	Dr. Wojtek J. Kowalczyk (LIACS)
2nd supervisor:	Dr. Frank W. Takes (LIACS)
3rd supervisor:	Dr. Kiamars Vafayi (External)

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Anti-Money Laundering (AML) practice tackles not only the illicit financial activities themselves but also the inhuman crimes behind, such as human trafficking. AML is essential to the safety and prosperity of our society. However, despite the substantial social meaning and the strict AML regulations, the result of AML practice is still not satisfying. Rule-based methods or other traditional methods are still dominating because of their relative effectiveness and explainability. However, rules are handcrafted, which involves many human efforts, yet they are fragile solutions; criminals can learn quickly to fool them. Meanwhile, the development in machine learning, specifically Graph Convolutional Networks (GCNs), has shown their promising applications to the problems that involve graph data. The AML data can be seen as a transaction graph where nodes represent accounts and edges represent transactions. Promising GCN solutions to AML are able to update models frequently and automatically; thus, significantly reduce human efforts. Yet, the research on applying GCNs to the AML problem is quite limited; the main reason is the data unavailability since transaction data is confidential.

In order to overcome this limitation, this thesis proposes an improved AML data simulator AMLSim-R, which incorporates realistic money laundering statistics to alleviate the data availability problem. This thesis considers the AML problem as a binary classification problem and investigates the effectiveness of Support Vector Machine, Logistic Regression, Decision Tree, Random Forest, XGBoost and three GCN variants on the AML dataset simulated by AMLSim-R. The thesis shows that classical methods especially XGBoost perform very well; the added value of GCN to the AML problem is limited.

Acknowledgements

I would like to express my deepest gratitude to my thesis supervisor Dr. Wojtek Kowalczyk for offering invaluable insight into my thesis project, and for all the enlightening discussion about life and the future. His kindness and encouragement also strongly backed me up in my study. Many thanks should also go to my second supervisor Dr. Frank Takes for his acute advice on my thesis.

I am also grateful to my external supervisor Dr. Kiamars Vafayi, who has encouraged me and been optimistic about my work throughout my internship, which gave me much confidence to proceed with my work. I also wish to thank Mirjan Ramrattan, who has always been supportive and considerate of me; her kindness gave me a soft landing in my career. Thanks also to all my colleagues for generously giving compliments and constructive criticism on my work.

I wish to express my sincere gratitude to Mei-Chen Lin for understanding my sensitive feelings and being supportive. I would like to extend my sincere thanks to Nelly Palacios for supporting me emotionally and academically. I thank them both for always trusting in me. Their consideration is a torch in my hard time.

I cannot begin to express my thanks to my parents and my brother, without whose support I would have never gone this far. I thank them for supporting me as much as they can, and for always being there to share my joy and sorrow.

And I thank myself, for having the courage to take this journey.

Contents

Chapter 1

Introduction

Anti-Money Laundering (AML) practice has the purpose of preventing financial infrastructure from being used by criminals to legalise their "dirty" money. Illicit money often, in the form of cash, comes from organised crimes that involve enormous profit, such as human trafficking and drug smuggling. It is estimated that 2.45 million people are suffering as victims of human trafficking as of 2005; yet this figure is not complete as only a small percentage of such crimes were identified (FATF, 2011). The estimated amount of money laundered is 2% to 5% of global Gross Domestic Product (GDP) (United Nations, 2021), which is $1.8 trillion to $4.4 trillion in 2019 (World Bank, 2021). Given the severity of the involved crimes and the tremendousness of the amount of money, AML is vital and necessary. Because successful AML practice can not only result in clues for fighting crimes but also cut down illicit money flows, thus reducing crimes.

In 2015, European Union (EU) renewed the directive (European Commission, 2015) on the prevention of money laundering and other financial criminal activities. The directive also urged EU state members to legislate against such crimes. The legislation compliance mainly goes to financial institutes, especially banks. They are obliged to conduct forensic analysis and to monitor transactions. A Suspicious Activity Report (SAR) has to be filed to Financial Intelligence Unit (FIU) if financial institutes detect any suspicious transaction. The penalty for failing the compliance is substantial: the Dutch bank ING was fined €775 million for allowing criminals to launder money through its network (Reuters, 2018); the US bank Goldman Sachs was fined $2.9 billion for money laundering related bribery (The Washington Post, 2020).

Despite the strict legislation and much effort dedicated to AML, it was estimated that only 1% of the illicit money was confiscated (Europol, 2017). There are two main reasons to explain why only a few money laundering activities are detected. The first is that some

banks fail to fulfil the regulatory compliance to make efforts to detect suspicious financial activities. The second reason lies in the highly sophisticated money laundering strategies (strong opponents) and the arduous nature of AML (difficulties in forensic analysis).

Organised criminal groups often have well-developed money laundering tactics. They might conduct international, inter-currency transactions over a long period to blur the source of the illicit money. This also relates to the forensic difficulties in AML. The shortage of discovered money laundering cases results in the lack of labelled transactions in financial systems, making AML methods hard to learn the patterns behind money laundering activities.

The ultimate goal of money laundering is explained by its name: laundering the dirty money clean, making the illicit money legitimate. To achieve this, money launderers use different strategies to hide the illicit essence of the money and disguise them as legitimate. Because of this, the research regarding anti-money laundering is not similar to most scientific researches that try to discover static patterns in nature. But much harder, AML tries to discover the dynamic patterns that are deliberately designed not to be found.

Machine learning methods have been successfully applied to credit fraud prevention and detection. However, the research regarding the application of machine learning methods to AML is quite limited. The main obstacle is the unavailability of labelled data for research in AML. Most of the previous work has focused on surveying classical machine learning methods that can be used for AML and on categorising them. The most relevant methods include classical classifiers like XGBoost and more advanced ones like Graph Convolutional Networks(GCN). This thesis aims at enriching the research on AML by addressing two main research gaps: the lack of datasets used for AML research and the application of novel algorithms to such datasets.

Regarding the data availability, the most reachable AML data source is synthetic data, while there are only a few and with low quality. Developing a scientifically usable synthetic data simulator is vital for future AML research development.

Regarding novel algorithm applications, this thesis focuses on two groups: GCNs and classical classifiers. The research question for GCNs is:

> *Are GCNs applicable to the AML problem? What influence their applicability?*

For classical classifiers:

> *How is the effectiveness of classical classifiers on the AML problem? Can they be improved by incorporating AML-specific information?*

To address these two research questions, this thesis proceeds and makes contributions as the following:

1. Developed an AML dataset simulator *AMLSim-R* (Realistic) (Chen, 2021) for the research community to use. It is built upon an existing data simulator AMLSim (Weber et al., 2018), incorporating realistic and robust statistics about money laundering and general transaction information to improve the data quality.
2. Verified the effectiveness of five classical classifiers on the highly unbalanced dataset. A metric, *top recall*, that aligns better with the interest of banks is used.
3. Presented the workflow of applying Graph Convolutional Networks (GCN) on AML and illustrated the key influencers of model performance.
4. Proposed a classification model that integrates graph embedding methods (node2vec) into XGBoost, achieving >0.99 top recall on synthetic data.

Thesis Structure The reminder of the thesis is structured as follows. Chapter 2 provides more details about the AML problem. Chapter 3 presents related work in AML topic from diversified directions. Chapter 4 scrutinises an existing AML data simulator throughly and describes the improvements made over. Chapter 5 investigates the effectiveness of classical methods. Chapter 6 shows the application of GCN to the AML problem; it also apply the findings in GCN into classical methods. Chapter 7 presents discussion and conclusions.

Chapter 2

Background

This chapter gives more details about the AML problem, including the typical money laundering process (Section 2.1), the arduous nature of AML (Section 2.2), the common practice in AML (Section 2.3) and the machine learning practice in AML (Section 2.4).

2.1 Money Laundering Process

Money laundering has the following definition (United Nations, 1998):

> *"the conversion or transfer of property, knowing that such property is derived from any offense(s), for the purpose of concealing or disguising the illicit origin of the property or of assisting any person who is involved in such offense(s) to evade the legal consequences of his actions".*

Money laundering can be roughly split into three phases. The first is *placement*, where illicit money is placed into the financial system by means of, for example, a large amount of small cash deposits. This work can be done by the so-called "money mules" who conduct such actions to earn commissions.

The second phase is *layering*. Once money enters the financial system, among other strategies, it can be transferred through different middlemen across different countries in different currencies, multiple times. The purpose of the layering actions is to make the origin of the money untraceable.

The last phase is called *integration*. This phase finally makes the illicit money look legitimate and available to criminals by, for example, making commercial investments.

A typical money laundering case can be as the following. An illegal drug cartel produces and sells its products on the black market and obtains a large amount of cash. The cartel then distributes the cash to a large amount of cartel members or other un-convicted ordinary people to deposit the cash into the financial system (*placement*). Then they transfer the money to different middlemen and the middlemen again transfer money to other middlemen at different levels (*layering*). Once the origin of the illicit money is untraceable, the money can be used to make investments such as opening a beauty salon (*integration*). With such a salon, future illicit money can be mixed with licit income to be laundered clean: a more easier follow-up money laundering method is settled.

2.2 Arduous Nature of AML

Society has been developing rapidly towards the cashless era. Daily transactions are being conducted in digital forms, such as mobile payment and peer-to-peer (P2P) transfers. This development explosively increases the usage of financial infrastructure and the number of transactions, which provide a thick disguise for money laundering activities. In 2019, Dutch Financial Intelligence Unit (FIU) identified 39,544 suspicious transactions (Dutch Government, 2019). Even if this number is only 1% of all suspicious transactions, the total number of suspicious transactions is still, comparing to the number of all transactions, a needle in a haystack.

Technically, this characteristic brings obstacles to forensic analysis. The first is data imbalance. The AML problem focuses on two labels of the data: suspicious and normal. (Banks do not have the ability nor the power to adjudicate whether a detected suspicious activity is truly a crime or not; thus, from the view of banks, suspicious and normal are the only possible labels.) The numbers of the two labels of either accounts and transactions are highly skewed, making the latent patterns behind the suspicious ones difficult to learn.

The other issue related to data imbalance is that only a few transactions are labelled. According to the 1%-estimation, the majority of suspicious activities are not detected nor labelled. The lack of labelled data, first, deteriorate the skewness of the data. Second, the purpose of AML is not only to detect the suspicious activities that follow the patterns found in the 1% known suspicious accounts; but also the ones that are not discovered yet. This purpose would be difficult to achieve, giving the lack of guidance in detection methods.

The patterns used in money laundering activities also keep evolving; criminals are innovative. From the view of banks, suspicious activity detection is about finding the patterns of criminal. This means banks are always in the chasing role, which is the passive side of the battle. Moreover, criminals are able to adjust their methods to break the rules banks found.

For a simple example, banks might monitor the amount of money in each transaction and record the ones that exceed €10,000 for further manual investigation. Breaking money laundering transactions into multiple small transactions might bypass this scheme.

Illegal financial activities also take advantage of the growing privacy-protection awareness. Money laundering transactions can occur across different banks; however, sharing client information among other banks are legally forbidden. Illicit money flows that involves multiple banks then are only partially visible to each individual bank, which makes suspicious activity detection for this kind more difficult.

2.3 AML In Practice

The typical method used in banks is transaction monitoring, around which five key steps are built (Weber et al., 2018). (1) Establishing a responsible compliance department; (2) following the Know Your Client (KYC) guideline to screen the profiles of the clients; (3) monitoring (computer-assisted) transactions and account activities, recording the suspicious ones; (4) manual investigating the recorded transactions and accounts; (5) filing Suspicious Activity Reports (SARs) to authorities and conducting corresponding actions on the suspicious transactions and accounts.

Under the transaction monitoring system, rule-based "if-then-else" triggers might be less efficient as money launderer keep trying to find loopholes in the rules and keep changing their strategies. Another type of triggers may have a form of applying classification algorithms. Logistic Regression (Nelder and Wedderburn, 1972) and Random Forest (Svetnik et al., 2003), among others, are prevalent models because of their relative effectiveness and explainability. Especially the explainability, it is essential for banks as they have to understand and explain why certain decisions are made, e.g. why the model declines a client's loan request; why the model classifies a particular transaction as a suspicious transaction. However, chasing explainable methods is a double-edged sword. The negative side is that human interpretable patterns are unfortunately very limited, further resulting in limited classification capability in models. Given the estimation that only 1% of the illicit money was detected, most patterns are not known nor explainable, at least to banks. One solution is to use more complicated methods, for example, machine learning methods, to explore the unknown space.

2.4 Deep Learning in AML

Deep learning has already had financial applications well developed and studied on credit fraud prevention and detection (Khandani et al., 2010, Perols, 2011, Butaru et al., 2016, Abdallah et al., 2016). However, research on deep learning-based AML methods is limited. Most of them are of survey nature that summarises the classical machine learning methods and categorise them (Chen et al., 2018, Rohit and Patel, 2015, Salehi et al., 2017). The most prominent reason for the limitation is the shortage of public AML datasets.

The complication in AML datasets, comparing to fraud detection datasets, lies in two aspects. The goal of fraud detection is to identify malicious transactions that steal money from banks or their clients. Once fraud transactions occur, clients often actively report such events. Therefore, datasets consist of transactions that serve the purpose of fraud detection is complete in terms of the quality of labels. Banks can assume that transactions that are not reported as fraud are normal transactions. However, this is not the case for AML datasets, where only a small number of money laundering transactions are filed as SARs. Yet even SARs can be unjustified. The second aspect is duration. Fraud transactions often can be reported within a short time and involve only a single transaction. Money laundering transactions, by contrast, involves multiple transactions (three phases) and last a long time. And banks may not conduct a proper practice to trace them as they do not have financial incentives to do so. The situation in fraud detection is different since banks have to compensate for the loss, so they are more motivated. Both characteristics impede the availability of AML datasets with good quality. Another reason worth mentioning is that transaction data is private data that is protected by regulations like General Data Protection Regulation (GDPR) (European Commission, 2016).

Therefore, AML research is done alternatively. The first strategy is to use **synthetic data** that incorporates known AML patterns. Graph Convolutional Networks (GCNs) (Kipf and Welling, 2016) is one of the tools that is applied to *synthetic transaction* data that is generated by a simulator called AMLSim(Weber et al., 2018). Applying classification methods on *synthetic account information* data is another direction (Lopez-Rojas and Axelsson, 2012). An alternative is to use **real data from banks** but under certain conditions. Suspicious transactions have different alert stages. The differences between the models trained on different stage are studied on real transaction data, which came from banks under usage agreements (Jullum et al., 2020). Graph analysis features extracted from real data was also used for AML by bank employees (Sangers et al., 2019). Another option is to work on **de-centralised financial system**, where transactions are freely open. The AML application of GCNs can be extended to bitcoin transaction graphs (Weber et al., 2019, Hu et al., 2019).

Chapter 3

Related Work

To address the challenges in the AML problem described in Chapter 2, attempts from different directions are made. This chapter summarises the previous work related to AML in the direction of datasets (Section 3.1), Social Network Analysis (Section 3.2), Graph Convolutional Networks (3.3) and Federated Learning (Section 3.4).

3.1 AML Datasets

To tackle the problem of lacking public money transaction datasets, *PaySim* (Lopez-Rojas et al., 2016) was proposed with the purpose of simulating mobile money transactions. It categorised transactions into five types: cash-in, cash-out, debit, payment and transfer. Each type of transactions was simulated according to statistical analysis on a large real dataset. The simulation consisted of three stages: (1) initiation: loading parameters; (2) execution: generating clients and transactions; (3) finalisation: producing log file and final simulated data. The resulting datasets were evaluated by the Sum of Square Error (SSE) between the real dataset reference and themselves. It was argued that PaySim was ready to supersede real financial data (when they are not accessible) for any research.

AMLSim (Weber et al., 2018) was built on PaySim with typical known money laundering patterns integrated. Before applying the transaction simulation, AMLSim constructed a *connection graph* where nodes were accounts and edges were connections, on which future transactions were simulated. AMLSim can be seen as an extension of PaySim that predefines graph structures.

Another problem relating to AML datasets is their imbalanced nature. Two directions of data sampling can be used to balance the data: over-sampling and under-sampling. Over-sampling methods increase the minority rate by sampling them with replacement in addition or creating synthetic minority data points (Chawla et al., 2002). Under-sampling methods, by contrast, keep the minorities intact while randomly selecting a proportion of majorities to achieve the goal of increasing the minority rate. Both strategies showed their effectiveness on the AML dataset from a U.S. financial institute (Zhang and Trubey, 2019). The results showed that the two prevalent classifiers Random Forest and Support Vector Machine (SVM) are able to gain from the minority ratio increases.

To tackle the problem of data unavailability in AML, this thesis proposes a new version of data simulator *AMLSim-R* (Realistic) for research communities to use or to improve further.

3.2 Social Network Analysis

AML datasets consist of accounts and transactions, which can be easily presented as nodes and edges in graphs (networks). Social Network Analysis (SNA) methods (Wasserman et al., 1994) are then nature options to tackle the AML problem.

SNA metrics can be used as feature indicators for bank accounts. In the factoring business, factors purchase invoices from sellers and redeem them to debtors. The factoring business has high risk of money laundering as invoices might just be disguises and no real goods or service is delivered. Such a business is studied on a real dataset from an Italian bank (Colladon and Remondi, 2017). The researchers constructed a transaction graph, an economic sector graph, a geographical area graph and a tacit link graph from the dataset. They applied SNA metrics, e.g. degree centrality (De Nooy et al., 2018) and network constraints (Burt, 2009), on the graphs to investigate the relationship between the metrics to high-risk accounts. High-risk accounts were defined by that if the owner of the accounts were involved in AML investigations or trials in the past three years. The results showed that SNA metrics were essential indicators for high-risk accounts. Degree centrality, betweenness centrality and network constraints appeared to be significantly positively correlated with high risk.

SNA also played a role in directly assigning money laundering roles (e.g. organisers and guardians) for nodes in the graph (Drężewski et al., 2015). SNA methods (e.g. centrality measures and network clustering) were applied to the relational data extracted from bank statements and National Court Register. Each node was assigned to a role according to the scores calculated from their SNA method scores. Results showed that such a role assigning algorithm is more efficient than the interconnection analysis.

Clustering is another SNA method relevant to AML. It is an unsupervised learning method that groups nodes into different clusters, where similar nodes are grouped together in the same cluster. In an AML graph with n nodes, the feature space of the graph can be mapped into $n + 2$ dimensional Euclidean space where the extra 2 dimensions are *transaction* and *time* (Zhang et al., 2003). The timeline can be discretised into different time instance, and the transactions can be represented by a scalar indication that aggregated money amount or transaction frequency. Therefore, the transaction dimension is collapsed into a point in the timeline axis. The accounts that have the same transaction scalar feature are counted together to form a histogram. Thus, the clustering problem is transformed into a histogram segmentation problem (Jain et al., 1995). This method performs in linear time to timeline space, which reduces the time complexity comparing to the prevalent clustering method *K-means* (Lloyd, 1982).

3.3 Graph Convolutional Networks

Since the introduction of Graph Neural Networks (GNNs) (Wu et al., 2020), they have been widely used in a variety of fields for a variety of tasks. GNNs are separated into two classes: GCNs (message passing based) and WL-GNNs (Weisfeiler-Lehman graph kernel based (Shervashidze et al., 2011)). A benchmark infrastructure is designed to fairly evaluate the performance of different GNN structures on different tasks (Dwivedi et al., 2020). The most relevant task to AML is node classification. It was concluded that directional GCNs that take advantage of attention (Veličković et al., 2017) and gating (Bresson and Laurent, 2017) mechanisms outperform un-directional ones.

GCNs were also directly used as AML tools on bitcoin transaction graphs (Weber et al., 2019). The dataset is facilitated with rich information: 166 features for each node, where 94 are local information about the node itself and 72 are aggregated information from its 1-hop neighbours. In this binary classification problem, the results showed that GCN outperforms Logistic Regression yet inferior to Random Forest and Multilayer Perceptrons. GCNs were also used as node embedding methods whose output is concatenated with the original node features. With such enhanced features, the performance of all the other three classifiers improved.

This thesis enriches the currently limited research on the application of GCNs to the AML problem by presenting the workflow of using GCNs as binary classifiers on AML and investigating the key influencers of model performance.

3.4 Federated Learning

Federated Learning (Konečnỳ et al., 2016, Yang et al., 2019) was proposed with the purpose of fusing data from different organisations together to present more available data to machine learning models, yet not leaking data to other organisations. Multiple organisations train a centralised model yet keep their own data distributed. Each organisation updates the model weights on their own data and communicate the updates with a third-party server, which is not able to infer original data from the updates. The third-party server collects the updates and produces a new model. The performance of federated learning models is expected to be very close to that of models trained traditionally on all data collected in one place but better than any model that is trained on the individual partial data from one organisation.

In the AML scenario, with federated learning methods, banks are able to collectively train a model while still keeping their client data secret from other banks and complying with the privacy protection regulations. A larger transaction graph can be collectively constructed, revealing more information about cross-bank transactions. PageRank (Page et al., 1999) on transaction graphs has shown its added value in detecting fraudulent transactions (Molloy et al., 2016). Three major Dutch banks calculated PageRank for every node in a collective transaction graph that consists of their data in a federated learning manner. Yet, each bank only knows the PageRank of the accounts in their own sub-graph (Sangers et al., 2019). In a large transaction graph, more information is feed into PageRank calculations to produce more comprehensive scores, comparing to the calculations on the sub-graphs. Such an improvement can further benefit the downstream suspicious activity detection.

Chapter 4

Data

This chapter describes the AML dataset this thesis uses. It proposes an improved AML data simulator version *AMLSim-R* by scrutinising the public data simulator AMLSim (Weber et al., 2018) and presenting improvements over AMLSim.

Specifically, this chapter is structured as follows: Section 4.1 depicts the transaction data in the AML context. Section 4.2 describes how AMLSim simulates transaction data in detail. Section 4.3 discusses the quality of the public datasets AMLSim provided. Section 4.4 describes improvements that can be made over the public datasets and presents the final dataset that will be used for downstream analysis.

4.1 Transaction Data in Anti-Money Laundering

A transaction dataset consists of records of transactions, each of which involves a *sender* account and a *beneficiary* account, the amount of money transferred and the timestamp at which the transfer occurred, along with other attributes.

In an AML transaction dataset, most transactions are normal (legitimate) financial activities, for example, grocery shopping, salary payment. Meanwhile, a small number of transactions are money laundering transactions. By these transactions, criminals try to use the financial infrastructure to legalise their illegal income by imitating the behaviour of normal transactions.

Several empirical money laundering transaction patterns have been discovered (Weber et al., 2018), such as *fan_in* (Figure 4.1) where multiple senders transfer money to a single

beneficiary; *fan_out* (Figure 4.2) where a single sender transfers money to multiple beneficiaries; *cycle* (Figure 4.3) where a set of accounts send money to other accounts while receiving from the another, forming a cycle. For an AML transaction data simulator, therefore, it is essential to include these *suspicious patterns* that are empirically observed from real money laundering transactions.

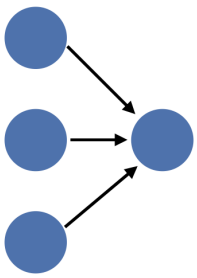


Figure 4.1: fan_in

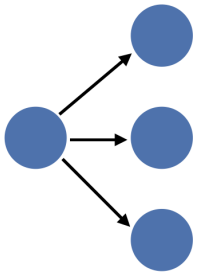


Figure 4.2: fan_out

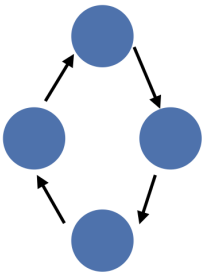


Figure 4.3: cycle

4.2 AMLSim Simulation Process

The simulation process of AMLSim involves two phases. The first phase constructs a *connection graph* where nodes represent bank accounts, and edges represent connections. Transactions can only occur between accounts that have a connection between them. The second phase simulates transactions pursuant to the connection graph. After the second phase, the connection graph is converted to a *transaction graph*, where edges are transformed from connections to true transactions. The two phases are elaborated in Section 4.2.1 and 4.2.2 respectively.

4.2.1 Graph Construction Phase

To generate the connection graph, AMLSim mainly takes the following three input configurations:

1. **Accounts**: It defines the number of accounts and descriptive attributes for each account, for example, initial deposits.
2. **Arbitrary Degree Distribution**: It defines the in-degree and out-degree distributions of the connection graph.

3. **Suspicious Patterns**: It defines the number of suspicious patterns that should be generated, and within each pattern, the number of accounts and the amount of money involved.

AMLSim first generates all nodes according to the **Accounts** configuration. It then adds base connections (edges) pursuant to the pre-defined **Arbitrary Degree Distribution**: the source node and the target node of a connection are randomly selected. After all connections are generated, the simulator identifies *hub* accounts whose degrees exceed a pre-defined threshold. Hub accounts are candidates for suspicious accounts.

The next step constructs an additional *pattern graph* for each suspicious pattern configured in **Suspicious Patterns**. Every suspicious pattern has a *main account*. For example, in a *fan_in* pattern, multiple accounts transfer money to a single beneficiary; thus, the single beneficiary is the main account in this suspicious pattern. Those main accounts are selected from hubs. In a suspicious pattern, not only the main account but all accounts that are involved in the suspicious patterns are considered suspicious accounts. Accounts will be removed from the candidate sets for other suspicious pattern constructions once selected for one pattern.

The nodes in each pattern graph is a sub-set of the connection graph, but the edges are new. The new edges are generated according to the characteristic of their corresponding suspicious patterns. Those new edges are attached with a money range that is configured in **Suspicious Patterns**. The range will be taken into account when generating true transactions in the second phase, in contrast to the money amount generation for normal transactions, where no individual ranges are given but all normal transactions share a large range. After construction, each pattern graph will be added to the existing connection graph. In other words, the newly generated suspicious connections (edges) will be added to the existing innocent connection graph. Transactions derived from those suspicious connections are considered to be suspicious transactions.

4.2.2 Transaction Simulation Phase

In the second phase, the simulator simulates transactions according to the connection graph. Unlike suspicious connections that are generated with money ranges specified, base connections do not have such specific ranges. One general range is applied to all transactions that are derived from base connections.

The transaction simulation is broken into steps, each step represents for one day. In each step, a number of normal transactions and suspicious transactions are generated randomly. The total numbers of suspicious transactions is consistent with the configuration in **Suspicious Patterns**; The number of normal transactions depends on the pre-defined the number of steps (days).

The output of the simulator consists of four sets.

1. **All accounts**: It lists all accounts in the dataset. Every account is identified by a unique ID along with other descriptive attributes.
2. **All transactions**: It lists all transactions in the dataset. Every transaction has a source account, a target account, the amount of money transferred, the timestamp and other descriptive attributes.
3. **Suspicious accounts**: It shows suspicious accounts. It is a sub-set of **All accounts**.
4. **Suspicious transactions**: It shows suspicious transactions. It is a sub-set of **all transactions.**

4.3 Public AMLSim dataset

AMLSim authors published five AMLSim datasets with different sizes. This section investigates the largest two of them and discusses the improvements that could be further made.

Each dataset comes with three files, missing *suspicious accounts* mentioned in Section 4.2. But suspicious accounts can be inferred from *suspicious transactions*, in which the accounts involved are suspicious; and also from *All accounts*, in which the accounts that are flagged as suspicious are suspicious accounts.

The first quality flaw in both datasets is that the numbers of suspicious accounts extracted from these two files are not equal: some accounts are flagged as suspicious in *All accounts* but are not involved in any suspicious transactions. To handle this inconsistency, this section considers the accounts that are truly involved in suspicious transactions to be suspicious accounts. Table 4.1 summarises relevant statistics of the datasets.

Table 4.1: Numbers of (suspicious) accounts and transactions in the two public datasets: p100K and p1M. "Sus" stands for "Suspicious".

dataset Name	p100K			p1M		
	Total	Sus	Sus Ratio	Total	Sus	Sus Ratio
Accounts	100,000	16,470	16.47%	1,000,000	160,753	16.08%
Transactions	12,476,012	17,052	00.14%	124,703,185	162,937	00.13%

The number part of the dataset names indicates the number of accounts in the datasets; the prefix "p" stands for "public (dataset)". Firstly the table shows that, in both datasets, the number of transactions is roughly 120 times the number of accounts. Secondly, the

numbers of suspicious accounts and suspicious transactions are roughly equal. According to the simulation process described in Section 4.2, however, the number of suspicious transactions should not be more than that of suspicious accounts, though this could happen in reality.

In the three suspicious patterns mentioned in Section 4.1, the number of suspicious accounts is always equal to or one more than that of suspicious transactions. In a *cycle* pattern, they are the same since the numbers of nodes and edges are the same in a cycle graph. In a *fan_in* or *fan_out* pattern, the number of nodes is one more than that of edges. Therefore, the two numbers should be similar and the number of suspicious accounts should be slightly greater than that of suspicious transactions. But the table shows otherwise. It is because several suspicious accounts are involved in multiple suspicious patterns, which should not happen according to the criteria of generation. This is the second inconsistency in the public datasets.

A major flaw in the datasets is the unrealistic ratios of suspicious accounts and suspicious transactions. In both of the datasets, the suspicious accounts ratio is 16%, which is unrealistically high. In real life, this ratio of suspicious accounts is often smaller than 1%. Such a high suspicious ratio in the public datasets undermines the speciality of highly skewed class imbalance in the AML problem. Therefore, both of the datasets are not suitable to be used in the further classification problem.

Another major flaw lies in the distribution of the amount of money in suspicious transactions. Figure 4.4 shows the distributions of the amount of money transferred in suspicious transactions in dataset p100K and p1M, respectively.

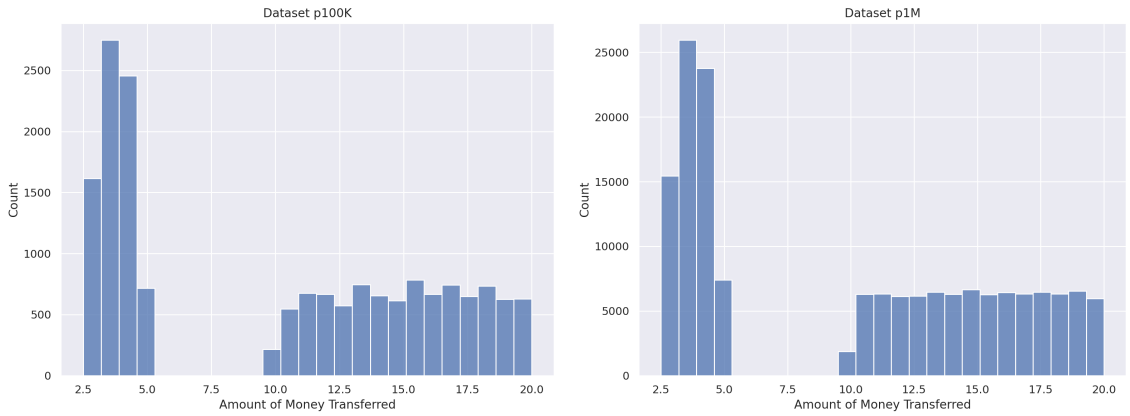


Figure 4.4: Distribution of the amount of money in suspicious transactions in dataset p100K (left) and p1M (right).

The money transferred in suspicious transactions falls into the range of $[3, 20]$. Most of the amounts are in the range of $[3, 5]$ and uniformly distributed on other amounts except that no suspicious transactions transfer money within the range of $[6, 10]$. Money laundering activities normally involve relatively large amounts of money (Dutch Government, 2019), but the maximum amount involved in the suspicious transactions in these two datasets is only 20. Although "20" doesn't have to be a specific currency, e.g. 20 Euro, but a scale; it is still at the lower level of all transactions since the money transferred in all transactions falls into the range from 1 to 21,474,837. Therefore, the distribution of the amount of money in suspicious transactions is problematic.

Other notable flaws include that for *fan_in* and *fan_out*, transactions in the same suspicious patterns always transfer the same amount of money. The simulator simply divides the total amount of money (configured in Suspicious Patterns configuration) by the number of accounts in that pattern and evenly assigns the value to all transactions. Such a practice firstly over-simplifies the real situation and secondly might be easy to be detected.

To summarise, the public datasets have the following major flows, which substantially affect the quality of the datasets:

1. The numbers of suspicious accounts in different output files are inconsistent.
2. The suspicious ratios of both accounts and transactions are unrealistically high.
3. The distribution of the amount of money in suspicious transactions is unrealistic, and the amounts only gather at a range with very low amounts.
4. The strategy that evenly assigns money to suspicious transactions over-simplifies the real money laundering situation.

Given the discoveries in this section, the quality of the published AMLSim datasets is not sufficient for downstream analysis. Thus, this work will improve all the discovered problems in the dataset and generate its own datasets for further analysis.

4.4 Dataset Quality Improvements

The simulator is highly parametrised. The three configurations discussed in Section 4.2 combines with a *general configuration* can produce different sizes of transaction datasets with different characteristics. This section discusses the parameter settings and the improvements over the default simulator in terms of suspicious transaction generation (Section 4.4.1), base transaction generation (Section 4.4.2) and general settings (Section 4.4.3).

4.4.1 Suspicious Transaction Generation

European Union published an overview report (Eurostat, 2013) about money laundering in Europe, and Dutch Financial Intelligence Unit (FIU) also delivered annual reports (Dutch Government, 2019, 2018, 2017) regarding the suspicious financial activities in the Netherlands. These official reports provide relatively realistic statistics, which can be used as the parameters for the data simulator.

Table 4.2 summarised the number of cases (patterns) and transactions that were identified as suspicious by Dutch FIU in the calendar year 2017, 2018 and 2019. In these three years, the minimum number of transactions per case is 7 and the maximum is 14. Given that they are averages already, a slightly moderate range, $[5, 15]$, for the number of transactions involved in a suspicious pattern will be used in the dataset generation process. The simulator does not allow to parameterise the number of suspicious transactions per suspicious pattern but allows the number of suspicious accounts. As illustrated in Section 4.3 that the number of suspicious transactions and accounts are roughly the same, we will set the number of suspicious accounts per suspicious pattern to $[5, 15]$.

Table 4.2: Numbers of suspicious cases and transactions in the Netherlands from 2017 to 2019. "AvgTxnPerCase" means "the average number of transactions per case"; the numbers are rounded to integers.

Year	transactions	cases	AvgTxnPerCase
2017	40,546	2,832	14
2018	57,950	8,514	7
2019	39,544	5,302	7
Average	46,013	5,549	8

Another critical configuration for constructing an AML transaction dataset is the distribution of the amount of money involved in suspicious transactions. That is, how many suspicious transactions involve how much money. For each money range, Table 4.3 summarises the percentages of the number of transactions and the amount of money involved.

Table 4.3: Percentages of the number of suspicious transactions and the amount involved in different amount ranges from 2017 to 2019. Due to rounding differences, the percentages do not add up to 100% precisely.

	2017		2018		2019	
Amounts	pct_num	pct_amt	pct_num	pct_amt	pct_num	pct_amt
€0-€10K	86%	3%	79%	3%	72%	2%
€10K-€100K	10%	14%	16%	18%	21%	19%
€100K-€1M	4%	35%	4%	38%	6%	36%
€1M-€5M	1%	48%	1%	42%	1%	43%
Total	100%	100%	100%	100%	100%	100%

Note that comparing to the original tables in the annual reports, two ranges: [€5M-€10M] and [€10M to more] are removed. Because these two ranges hold the majority (∼90%) of all suspicious money but only a few (<1%) transactions. This would make such transactions easy to be classified as suspicious regardless of their network structures. And the existence of these two ranges extremely skews the datasets. The percentages in Table 4.3 are re-calculated with the two ranges removed.

Table 4.3 shows that the majority of suspicious transactions involve less than €10K while the total value of them is no more than 3%. And for the range of [€1M-€5M], 1% transactions hold at least 42% of the total value. A power relation can be observed between the upper bound of each money range and its corresponding percentage of transaction numbers, as shown in Figure 4.5. In other words, as the money amount increase "powerly", the number of transactions decrease "powerly". Averages of the three years are shown in Table 4.4, which will be used to generate suspicious patterns.

Table 4.4: Averaged percentages of the number of suspicious transactions and the amount involved in different amount ranges. Due to rounding differences, the percentages do not add up to 100% precisely.

	Average	
Amounts	pct_num	pct_amt
€0-€10K	79%	3%
€10K-€100K	16%	17%
€100K-€1M	5%	36%
€1M-€5M	1%	44%
Total	100%	100%

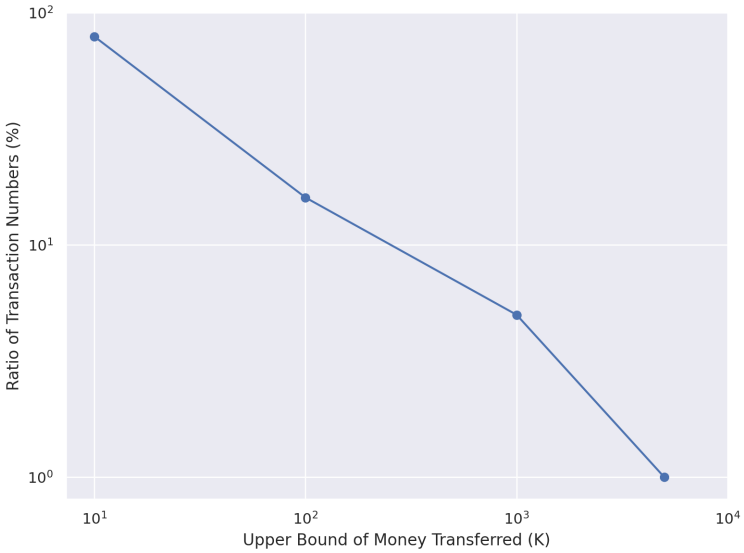


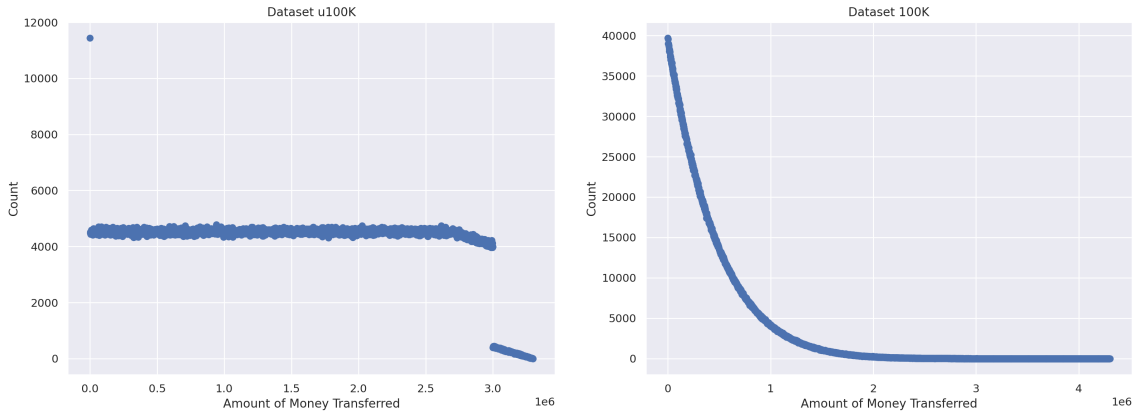
Figure 4.5: Relation between the ratio of transaction numbers and the upper bound of money amount.

4.4.2 Normal Transaction Generation

Base connections are generated without an individual range that limits the amount of money transferred for every transaction; they are randomly assigned a number from a large general range. AMLSim uses a **Uniform** randomness. Figure 4.6a shows the distribution of money transferred of all transactions in dataset u100K ("u" stands for uniform). The figure shows that the number of transactions is very similar regardless of the amount of money involved. For example, the number of transactions involving €5K to €10K is the same as that of €4M to €4.05M. Such a uniform distribution is rather problematic. The only exceptions are the outliers at the top-left and bottom-right sides: the bin represents the smallest range in the histogram contains the exceptionally high number of transactions; the bins represent the largest ranges contain the low number of transactions. These two area looks like reasonable, but the gaps between them and the main body are huge and rough, which are unrealistic.

To align with the distribution of the number of suspicious transactions discovered in Table 4.5, this chapter generates a dataset *r100K* ("r" stands for realistic) using a distribution that as the amount of money increases, the number of transactions decreases rapidly:

$$x' = T_{\min} + (1 - x^{\frac{1}{11}}) \cdot (T_{\max} - T_{\min}) \tag{4.1}$$



(a) u100K (b) r100K

Figure 4.6: Distribution of the amount of money in all transactions in dataset u100K and r100K.

where $T_{\max}$ and $T_{\min}$ are the maximum and the minimum of the base transaction range; x is a uniformly distributed float number between 0.0 and 1.0; x' is the resulting variable in the desired distribution, which is shown in Figure 4.6b.

4.4.3 General

Incorporating all the improvements, this chapter proposes an improved version of the data simulator: *AMLSim-R* (Realistic). Table 4.5 summarises its parameters used to generate the dataset *r100K*, which will be used for future analysis in the rest of the thesis.

Table 4.5: Parameter Settings for dataset r100K

General		dataset-Specified	
Param	Value	Param	Value
minAmt	1	numAcct	100K
maxAmt	5M	initDeposit	[5K, 5M]
txnWindow	720	numSusPatn	100
degThld	10	numAcctInSus	[5,15]
randomSeed	0	amtInSusDist	Table 4.4

The left side describes the general configurations. *minAmt* and *maxAmt* forms a range, from which base transactions randomly select the amount of money. *txnWindow* defines

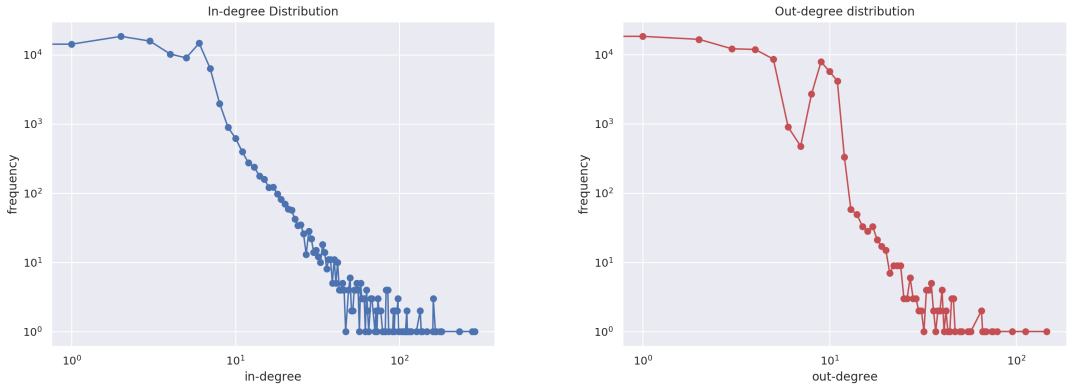
the time window of the dataset; all transactions are made within the time window. It is set to 720, which roughly equals 2 years. *degThld* is the degree threshold for accounts to be considered as hubs.

The right side shows the dataset-specified configurations included in the three configuration files mentioned in Section 4.2. *numAcct* is the number of accounts in the dataset; *initDeposit* determined the initial deposit range for all accounts; *numSusPatn* specifies the number of suspicious patterns (cases) that the dataset contains; *numAcctPerSus* is the number of accounts in one single suspicious pattern, it is set to $[5, 15]$ according to Table 4.2; *amtInSusDist* is the distribution of money involved in suspicious transactions, it is set according to Table 4.4.

Table 4.6 summarises the statistics of the dataset r100K. Suspicious ratios of both accounts and transactions are much lower than those in the public datasets. Figure 4.7a and Figure 4.7b show the in-degree and out-degree distribution of dataset r100K respectively. It can be seen from the log-log plots that both distributions follow the power law distribution.

Table 4.6: Numbers of (suspicious) accounts and transactions in dataset r100K.

	Total	Suspiciouss	Suspicious Ratio
Accounts	100,000	1,007	1.0e-2
Transactions	10,299,175	947	9.2e-5



(a) In-degree Distribution (b) Out-degree Distribution

Figure 4.7: In-degree and Out-degree distributions of dataset r100K

Figure 4.8 illustrates the number of transactions generated in each simulation step. The starting date is arbitrarily set to 2017-01-01; the number of steps (days) is 720.. Both base transactions and suspicious transactions are generated uniformly throughout the entire generation process. More than 10,000 base transactions and less than 10 suspicious transactions are generated in each step.

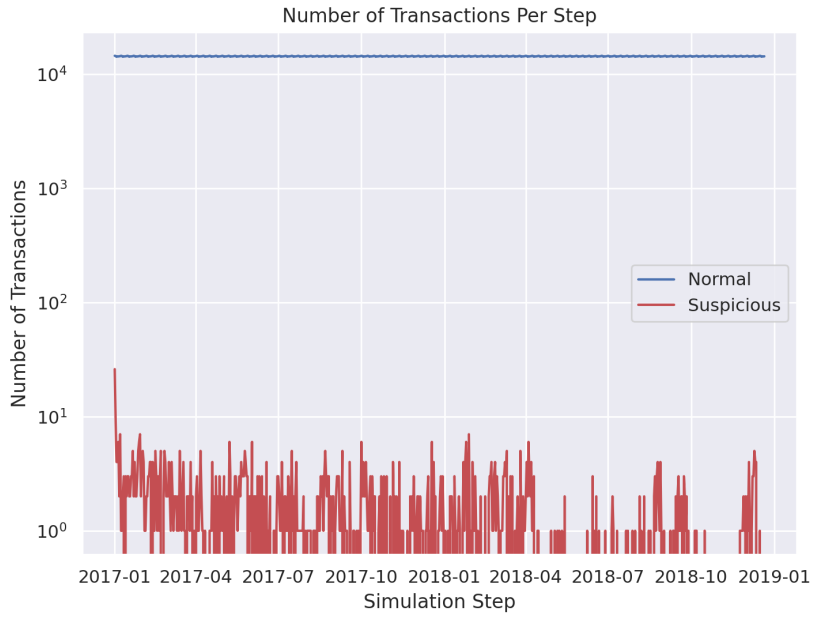


Figure 4.8: Distributions of the number of transactions generated over the timewindow.

4.5 Summary

This chapter scrutinises an AML transaction data simulator AMLSim, and the analytical results show that the data generated by AMLSim does not have sufficient quality for future analysis. This chapter addresses the flaws in the original data simulator and proposes AMLSim-R. The dataset *r100K* generated by AMLSim-R will be used for downstream analysis in the rest of the thesis.

Chapter 5

Classical Methods

This chapter investigates how classical classifiers perform on the AML problem. Section 5.1 formally defines the problem. To address the problem, Section 5.2 describes the three key components needed: the evaluation strategy, the metrics used to determine the performance and the classifiers evaluated. Section 5.3 shows the experimental setup in detail. Section 5.4 presents the results and discoveries. Section 5.5 summarises this chapter.

5.1 Problem Definition

A dataset $\mathcal{D}$ contains N_{txn} transactions between N_{acct} accounts. Accounts are labelled into two classes $L : \{sus, nor\}$ (stand for "suspicious" and "normal" respectively), the numbers of accounts of each label are N_{sus} and N_{nor} respectively. Transactions in the "sus" class are considered suspicious of money laundering; they are the primary interests of the AML problem yet this class is the minority in AML datasets. Transactions in the "nor" class are considered normal and legitimate transactions; this class is the majority in AML datasets. In the dataset this thesis uses, the ratio of $\frac{N_{sus}}{N_{sus}+N_{nor}}$ is roughly 1% as presented in Table 4.6, Chapter 4.

A classifier $\mathcal{C}$ predicts a label for each account:

$$L'_i = \mathcal{C}(a_i), i \in \{0, 1, ..., N_{acct}\} \tag{5.1}$$

where a_i is the i-th account in the dataset and L'_i is its predicted label. The problem is then: how well do classifiers' predictions (L'_i) align with their genuine labels (L_i) for all the accounts in the dataset. There are only two possible labels in the dataset and all accounts have labels; this is a supervised binary classification problem.

5.2 Methodology

This section describes the three salient parts that are needed to address the problem. Section 5.2.1 shows the evaluation strategy that defines how the classifiers will be evaluated; Section 5.2.2 describes the metrics for measuring the performance of the classifiers; Section 5.2.3 briefly describes each classifier this chapter uses.

5.2.1 Evaluation Strategy

The main goal of this chapter is to select the *best classifier* among the tested ones by evaluating their classification capabilities. This differentiates from selecting the *best model*. This chapter refers a classification method to a *classifier*, e.g. a random forest; refers *weights* that are trained under a combination of a certain classifier and a set of its corresponding hyper-parameters to a *model*, e.g. a random forest with the number of trees of 10 trained on r100K.

Selecting the best model requires the dataset to be split into a training set and a test set. Every classifier would be trained on the training set and yield its best model. All models are then evaluated according to their performances on the test set. In this way, however, a significant part (at least 20%) of the dataset would be held out from the training process, which would cause two major problems. The first one is that the size of the training set was reduced. Apart from the test set, a validation set would be further split from the rest of the dataset, which also reduces the size of the training set. Insufficient training data would produce an under-fitted model or over-fitted if the model is too complicated; either way would result in deficient generalisation capabilities. The second problem lies in the performance measuring criterion. The performance on the 20% of the dataset is not a robust signal for the generalisation capability of a model.

Selecting the best classifier, by contrast, differentiates selecting models from the basis. It does not require producing trained models for future use; the purpose of it is to evaluate classifiers' generalisation capabilities, giving a classification problem and a dataset. In other words, it evaluates models instead of building models. Therefore, the *nested Cross-Validation (nested CV)* (Cawley and Talbot, 2010) scheme will be used for this task. Figure 5.1 depicts such a scheme.

CV is a model evaluation strategy that splits a dataset into a training set and a validation set k times (also called k fold). In each fold, one k-th of the dataset is held out as the validation set; the rest of the dataset forms the training set. The validation sets do not overlap across all k folds while the training sets partially overlap. The purpose is to construct multiple datasets on which classifiers can be trained and evaluated. The resulting multiple evaluation scores then represent the estimation of the general classification

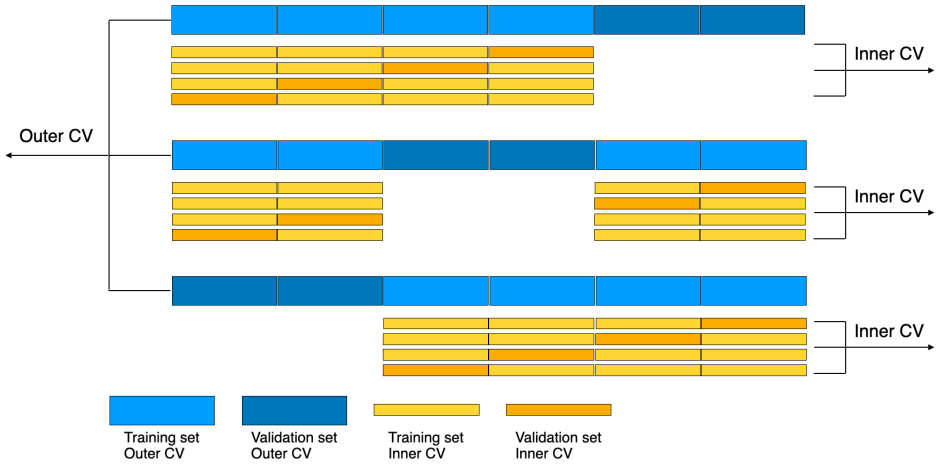


Figure 5.1: Nested cross validation scheme.

capability of a classifier.

Nested CV consists of two CV loops: the outer CV outputs k_{out} scores that come from k_{out}-fold training/validation set splits; the k_{out} scores are indicators of a classifier's general classification capabilities. The outer CV is conducted on the entire dataset; thus every part of the dataset is used for evaluation, bringing out more robust evaluation results. The inner CV is used to tune the hyper-parameters of a classifier, aiming at building the best model. The inner CV is conducted on the training sets of the training/validation set splits from the outer CV. The model with the highest inner CV score will be validated against the corresponding validation set split, the result of which is one of the k_{out} scores. Specifically, the nested CV scheme can be viewed from two directions.

From Top To Down The outer CV is the main character of the nested CV scheme. It provides k_{out} training/validation set splits, each of which is used to fit a model and validate a fitted model. The inner CV can be considered as a part of the fitting process. Alternatively, the inner CV could be replaced by a static training/validation set split: applying multiple hyper-parameter settings on the same training set and evaluating their performances on the same validation set. But for the sake of robustness, the hyper-parameter tuning process is also done by CV.

From Down To Top The inner CV is used to tune hyper-parameters. If it was the only CV, then within each fold, once the best hyper-parameter setting was selected according to the score on the validation set, there would be no more data to evaluate the

model's performance. The score on the validation set can not be considered as the model's performance because, in this way, the same data was used for both model building and model evaluating, which would give overly optimistic estimations. Therefore, besides the data that is used for tuning hyper-parameters, another unseen set of data should be held out for evaluating the winning hyper-parameter set. The held-out validation set and the hyper-parameter-tuning set forms a fold of the outer CV.

5.2.2 Metrics

To evaluate the performance of a classifier in a binary classification problem, four types of outcomes are essential.

1. True Positive (TP): this is the result when a classifier correctly predicts a sample to the positive class (the *suspicious* class, in the AML scenario); thus a target hit.
2. True Negative (TN): this is the result when a classifier correctly predicts a sample to the negative class (the *normal* class)
3. False Positive (FP): this is the result when a classifier incorrectly predicts a sample to the positive class; thus a false alarm.
4. False Negative (FN): this is the result when a classifier incorrectly predicts a sample to the negative class; thus a target miss.

A Variety of metrics derive from these four concepts (Provost and Kohavi, 1998): (1) *accuracy*, calculated as $\frac{TP+TN}{TP+TN+FP+FN}$, measures the proportion of correctly predicted samples for both positive and negative classes; (2) *recall*, calculated as $\frac{TP}{TP+FN}$, measures how many genuine positive samples are correctly predicted; (3) *precision*, calculated as $\frac{TP}{TP+FP}$, measures how many samples that are predicted as positive are genuine positive. Note that in this AML problem, the suspicious class is the positive class, and the normal is the negative.

The dataset has only two possible labels and is highly skewed; the ratio between suspicious accounts and normal accounts is 1%. Thus metrics that consider the correctness for both labels, like *accuracy*, are not suitable. For a binary classification problem with highly skewed data, classifiers could predict all samples as normal accounts and easily get high accuracy. This is not desired, however. Only correctly predicting suspicious accounts is the interest of the AML problem.

Classifiers do not directly classify a sample to a class; they predict the probabilities (or likelihood for some classifiers such as SVM) of a sample belonging to a class. For example, a classifier may predict that account A has 95% probability belonging to normal accounts and 5% to suspicious accounts; normally, classifiers classify a sample to the label that has

the highest probability. But in a skewed dataset, classifiers may be overwhelmed by features of normal accounts and hard to learn those of suspicious accounts due to the insufficiency of data points, which would result in the tendency of predicting high probabilities of being normal for most of the accounts. In this case, seldom accounts would be predicted as suspicious.

This thesis addresses this by only focusing on the class interested. That is, only focusing on the probabilities that classifiers predict accounts to be suspicious. Various discrimination thresholds can then be set to determine their predicted labels: accounts with probabilities higher than the threshold are labelled as suspicious, lower are normal, or the other way around. This can be formulated as follows:

$$[P_{i\_sus}, P_{i\_nor}] = \mathcal{C}(D_i), i \in \{0, 1, ..., N_{acct}\} \tag{5.2}$$

$$L'_i = \begin{cases} Suspicious & P_{i\_sus} > P_{thld} \\ Normal & P_{i\_sus} < P_{thld} \end{cases}, i \in \{0, 1, ..., N_{acct}\} \tag{5.3}$$

where N_{acct} is the number of accounts, D_i is the i-th account, P_{i_sus} and P_{i_nor} are probabilities that a classifier predicts for each of the classes; L'_i is the predicted label of i-th account, P_{thld} is a pre-defined threshold. Thresholds can be set to any value between 0% and 100%; different thresholds determine different numbers of correctly classified accounts. One strategy of defining thresholds is sorting all P_{i_sus}'s and then, from the minimum to the maximum (or from maximum to minimum), setting the threshold to values in the sequence in order. In this way, varying thresholds would produce monotonic performances in terms of the number of correctly predicted accounts. The following metric adapts this idea.

Receiver Operating Characteristic (ROC) Curve & Area Under Curve (AUC) ROC curves are illustrated by plotting the *true positive rate (TPR)* against the *false positive rate (FPR)*. True positive rate is the same as recall, calculated as $\frac{TP}{TP+FN}$ while false positive rate is calculated as $\frac{FP}{FP+TN}$. They can also be reformed as $\frac{TP}{N_{pos}}$ and $\frac{FP}{N_{neg}}$ in this AML problem. The denominators of both terms are constant regardless of how classifiers perform; what ROC measures then can be seen as: what is the cost (false positive) for achieving the benefit (true positive). When lowering the discrimination threshold, more accounts would be predicted as positive[1] while some of them are genuine negative thus increase the false positive rate. Effective classifiers should produce results that the number

[1]To make it simple for demonstrating, this work assumes samples with probability higher than the threshold are predicted as positive

of true positives is much higher than that of false positives when lowering the threshold. Another salient point about ROC is that it is data imbalance-proof. No matter the ratio between N_{pos} and N_{neg}, each of them is only used in one rate (either TPR or FPR). Thus, changing the data distribution will not change the shape of ROC curves significantly, giving the same classification capability. Metrics like precision, on the contrary, involve both the number of positives and negatives: TP is part of N_{pos} while FP is part of N_{neg} in its calculation $\frac{TP}{TP+FP}$; thus may produce misleading results in imbalanced datasets.

AUC measures the area under the curve; it equals the probability that a classifier ranks a randomly chosen positive sample higher than a randomly chosen negative sample (Fawcett, 2006). Thus the higher the AUC, the better the performance.

Top-recall Another metric used in this chapter is *top-recall*. It is not plausible for banks or any organisation to investigate every transaction in the financial system; only highly suspicious transactions will be further investigated. Therefore, closely looking at the performance of models on the *top accounts* will be more feasible and efficient. Top accounts are defined as the accounts that have relatively high probabilities of being suspicious.

Top-recall is calculated as the following: 1) sorting all accounts according to their probabilities of being suspicious; 2) assuming the top 20% accounts are predicted as suspicious by a model; 3) calculating the ratio between the number of correct predictions in the 20% accounts and the number of all genuine suspicious accounts in the dataset. From another point of view, top-recall measures how many genuine suspicious accounts are covered by the top accounts.

It's a common practice to present *precision* and *F1* scores along with recall scores in model performance evaluation. However, that two measurements are less meaningful in highly skewed datasets. If top precision scores were calculated, the maximum would be 0.05 ($\frac{1}{20}$). Because only 1% of the accounts in the dataset are suspicious, even if all of them were covered in the top 20% accounts, the top precision would be only 1/20, which could not fairly reflect the performance of the models. Since precision is part of the calculation of F1, F1 is not a suitable measurement either.

To summarise, For the two metrics, top-recall scores will be the main metric as it focuses on the performance of the top accounts as it better aligns with the interests of the AML problem. Although ROC curves and AUCs measure the performance of models on all accounts, which may give over-pessimistic results, they are able to visualise the classification capabilities of models; thus, they will be the supplementary metrics.

5.2.3 Classifiers

This chapter investigates five classical classifiers; their scikit-learn implementation (Pedregosa et al., 2011) is used in the experiments.

Support Vector Machine (SVM) In a binary classification problem, SVM (Cortes and Vapnik, 1995) tries to learn a hyperplane between the two classes to maximise the margins between the hyperplane and the two classes. SVM is a linear supervised learning model.

Logistic Regression Logistic Regression (Kleinbaum et al., 2002) uses a logistic function to model the independent input features to predict the probability of being one of the binary labels. Since logistic functions can be set to have output ranges of [0,1], they are well fit in binary classification problems. It learns its weights by maximising the likelihood between the predictions and ground truths. Logistic Regression is also a linear supervised learning model.

Decision Tree A Classification And Regression Tree (CART) (Breiman et al., 1984) recursively partitions the input feature space into a set of regions according to partition criteria until the data points in the regions reach a pre-defined threshold. All the resulting regions are then the leaf nodes of the tree. For a new data point that follows the tree partition criteria and ends up in leaf node l, it is predicted as the label that is the majority in node l.

Random Forest Random Forest (Ho, 1995) is an ensemble of bagged tress. *Bagging* is a technique that averages the predictions from models trained on multiple *boostrap* datasets. Bootstrap randomly draws B datasets from the training set with replacement, each of the B datasets has the same size as the training set. Therefore, Random Forest trains a classification tree on each one of the bootstrap datasets, resulting in B trees. To predict the label for a new data point, Random Forest aggregates the predictions from all B trees and predicts the label that has the most votes.

During the training, CART trees in Random Forest do not use all features to grow (split) the tree. Instead, for each tree, at each node, m features are randomly selected to be the candidates. Among the m features, the one that produces the best split (minimum error) is used to split the node into two child nodes. The trees keep growing until the number of data points that fall in the node is smaller than a pre-defined threshold. Therefore, Random Forest can be used to identify which features have more influence on its performance since important and decisive features will be selected to split the node more often.

XGBoost XGBoost (Chen and Guestrin, 2016) stands for *Extreme Gradient Boosting*, where "Extreme" emphasises its scalability, "Gradient Boosting" (Friedman, 2001) is its training strategy. The difference between XGBoost and Random Forest mainly lies in how they train their tree ensembles.

Unlike bagging, which is used in Random Forest, equally trains all the models within the ensemble; boosting trains models sequentially in a way that new models learn from the errors old models made thus newer models keep evolving themselves and finally form a powerful ensemble.

The XGBoost version used in this chapter adapts the *additive training* strategy. The output of the successor tree combines the outputs from all predecessors:

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) \tag{5.4}$$

where t is the t-th tree in line; f_k is the output of the k-th tree in line; x_i is the i-th data point in the dataset. Optimising the objective function that only involves the last tree will optimise all trees in the ensemble.

To summarise, this chapter will investigate the above five classical classification methods, two of which are linear methods while three of which are tree-based methods.

5.3 Experimental Setup

This section describes the experimental setup in detail. Section 5.3.1 shows the features the classifiers use; Section 5.3.2 shows how nested CV is set up.

5.3.1 Data Features

The data simulator outputs all simulated accounts and transactions, from which the following features are extracted for each account to train the classifiers.

1. Initial deposit: the initial deposit.
2. In/Out-Degree: the number of income/outcome transactions.
3. In/Out-NumAccts: the number of accounts with which the interested account makes income/outcome transactions. Note that *Degree* and *NumAccts* differ when one account makes multiple transactions with the interested account.
4. In/Out-MoneySum: the sum of money of all income/outcome transactions.
5. In/Out-MoneyAvg: the average of money of all income/outcome transactions.
6. In/Out-MoneyMin: the minumum of money of all income/outcome transactions.
7. In/Out-MoneyMax: the maximum of money of all income/outcome transactions.

8. In/Out-Interval: the average interval of chronologically adjacent income/outcome transactions.

The intervals are measured by a unit of *days*. If an account has income/outcome transaction fewer than two, thus no intervals, -1 will be assigned.

For each account, the above 15 features form the feature space of the dataset along with a label indicating whether it is a suspicious account. The three tree-based methods use the original feature space; the two linear methods use the standardised feature space since the original feature space is not concentrated enough (e.g. the amount of money in transactions vary from €1 to €5,000,000), which may make the linear models hard to converge. Features are standardised according to $X' = \frac{X-\mu}{\sigma}$, where X' is the standardised feature, X is the original feature, μ and σ being the mean and the standard deviation of feature X.

5.3.2 Nested Cross Validation

Stratified k-fold Both outer and inner CV use the *stratified k-fold* scheme with $k = 5$. This scheme preserves the ratio of samples between each class in all folds. In other words, it guarantees to allocate both suspicious account samples and normal account samples in each fold and maintain the ratio as it is in the entire dataset (suspicious/normal: 1%). This scheme is crucial as it ensures models in every fold have the opportunity to learn from features of both classes.

Randomised Search The inner CV is used to tune hyper-parameters. The tuning process applies a *randomised search* strategy. Every classifier provides a hyper-parameter space, which states its to-be-tuned hyper-parameters and their ranges as well as sample methods (e.g. uniformly sample, logarithmically uniformly sample). Every inner CV runs 100 iterations of randomised searches. The winner that has the highest average ROC-AUC score is validated against the validation set of the outer CV, the result of which forms the final generalisation capability estimation. The hyper-parameter space of all five classifiers are shown in Appendix Table A.1, A.2, A.3, A.4 and A.5 respectively.

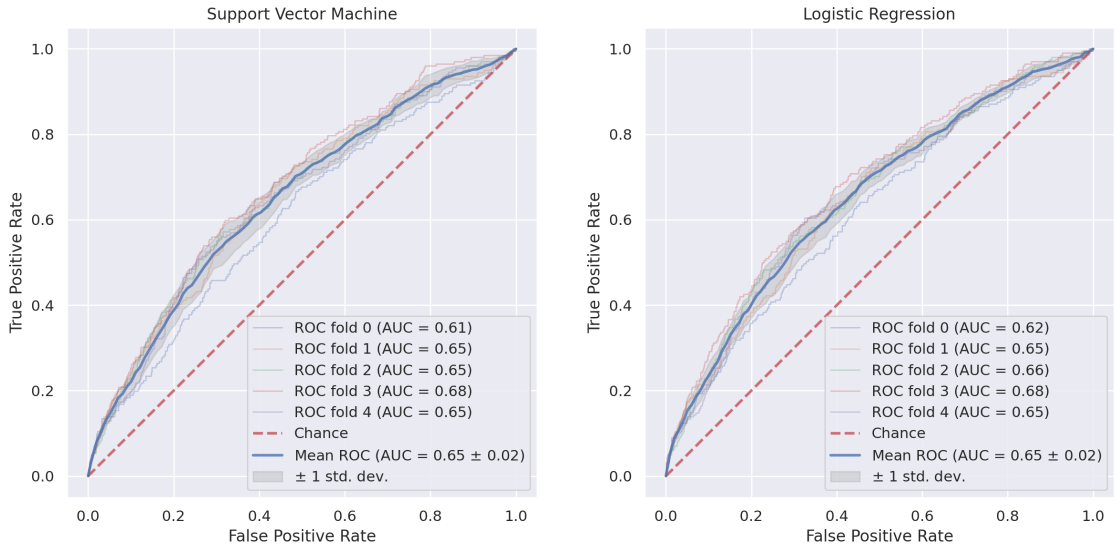
5.4 Experimental Results

This section discusses the experimental results, which are presented in two sections corresponding to the two metrics described in Section 5.2.2. Section 5.4.1 analyses the performance of the classifiers in terms of their ROC curves and AUC; Section 5.4.2 presents the top-recall scores.

5.4.1 ROC Curves and AUC

The ROC curves of the two linear models and the three tree-based models are summarised in Figure 5.2 and Figure 5.4 respectively. The red dash lines in plots stand for a trivial classifier that only randomly classifies data points to classes. The five lines with light colours in each plot represent the results from five cross-validation folds, the blue line is their mean, and the grey space is its ±1 standard deviation.

Linear Models From Figure 5.2 we can see that the Logistic Regression classifier and the SVM classifier produce similar results in terms of both ROC curve shapes and AUC scores. Their mean ROC curves are only slightly above the "chance" lines, indicating they outperform a random guesser yet not too much. The AUC scores of 0.65 indicate the same conclusion as a random guesser could achieve 0.50.



(a) Support Vector Machine

(b) Logistic Regression

Figure 5.2: ROC curves of 2 linear classifiers

To investigate why linear models do not perform well in this AML problem, t-distributed Stochastic Neighbour Embedding (tSNE) (Maaten and Hinton, 2008) is applied to the dataset to visualise it. tSNE maps data points with high-dimensional feature space into two or three-dimensional points in a coordinate for visualisation.

Figure 5.3 shows the result of applying tSNE to the AML dataset. All suspicious accounts are presented, but only 10% normal accounts are randomly sampled to be shown in the plot for visibility; otherwise the plot would be overwhelmed by blue points, which represent normal accounts. From the figure we can see that suspicious accounts are scattered among normal ones; thus, they are hard to be linearly separable, if not impossible. This figure shows the difficulty of this AML problem as no prominent outliers or patterns can be directly seen. More complicated models have to be applied to learn the latent patterns.

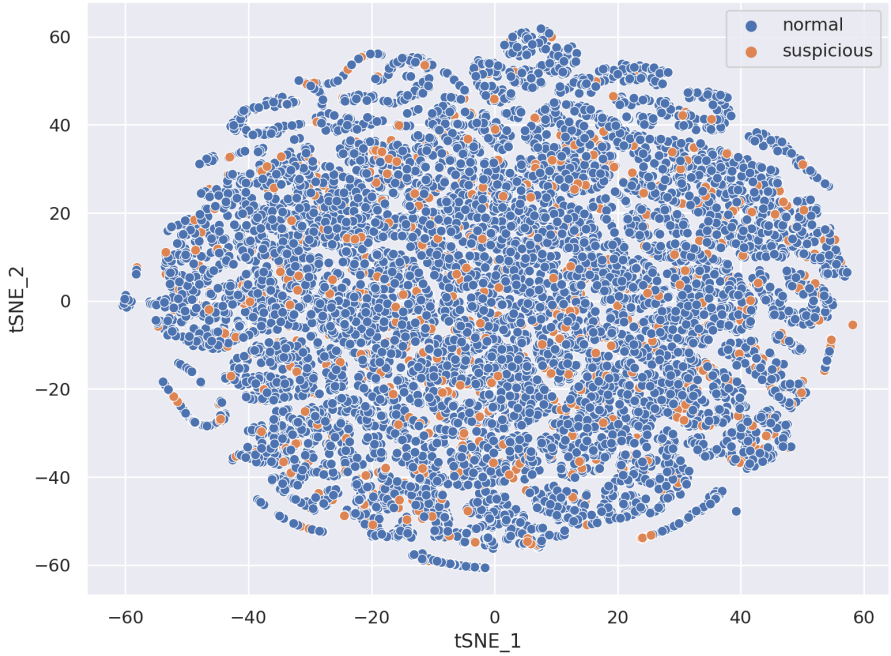
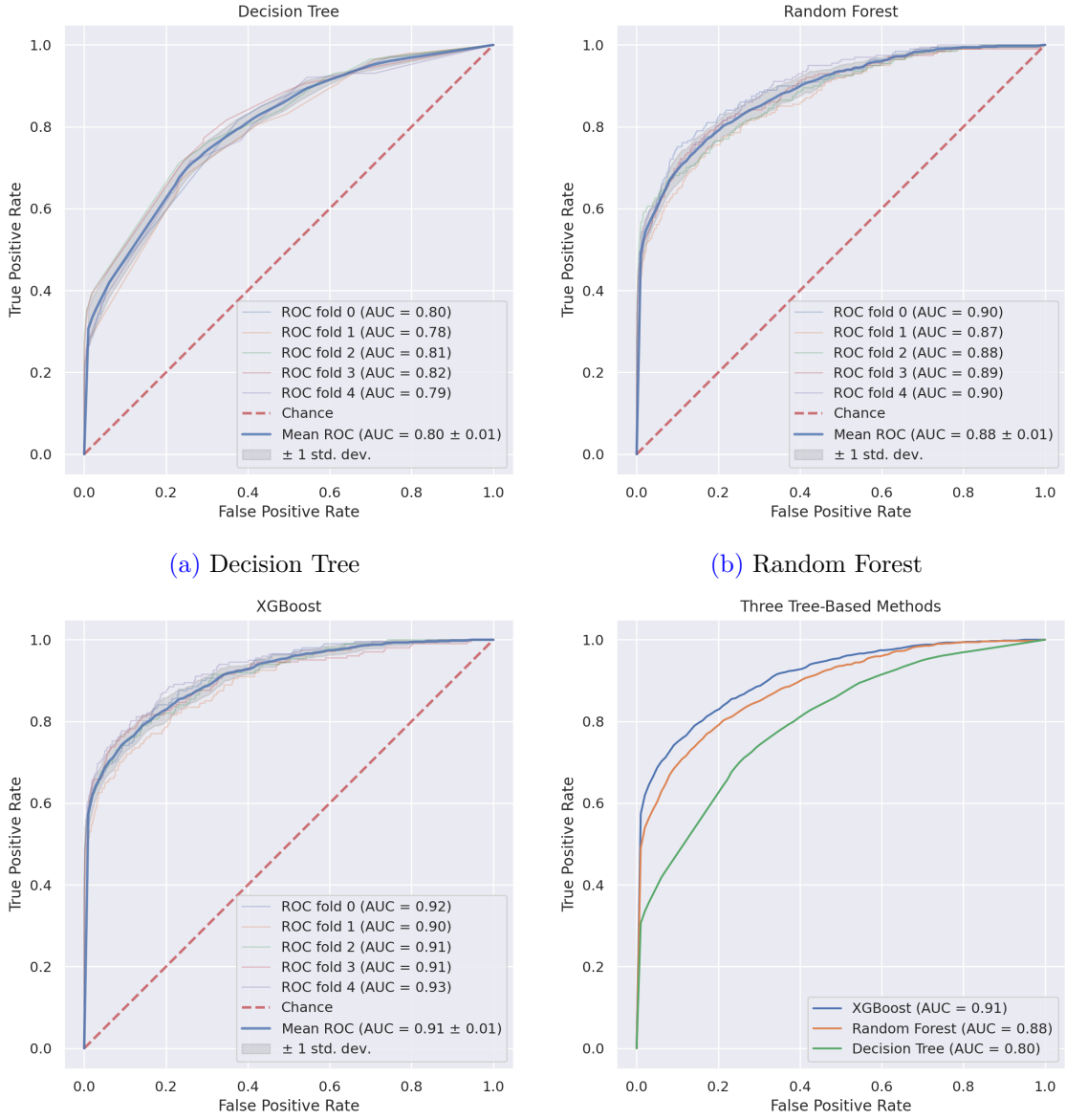


Figure 5.3: Visualisation of accounts using tSNE. All suspicious accounts and 10% randomly sampled normal accounts are shown.

Tree-based Models The performance of the three tree-based models are shown in Figure 5.4. We can see that all three classifiers yield ROC curves above the "chance" line and produce high AUC scores. The classifiers successfully exploit the information in the data and learn the capability of separating the two classes to a large extent. Each plot also shows small grey areas around the mean blue line, indicating low deviations, which then implicate the stability of each model.



(a) Decision Tree

(b) Random Forest

(c) XGBoost

(d) Three Tree-based Classifiers

Figure 5.4: ROC curves of the three tree-based classifiers.

The mean ROC curves in the three plots are summarised in Figure 5.4d for comparison. The most worth noticing part is the area near the y axis. All three curves are very steep alongside the y axis: the true positive rates increase drastically while false positive rates barely increase. As the threshold lowers, more data points are predicted as positive; the steep curves show that those newly positive predictions are genuine positives. All three classifiers, therefore, are able to gather genuine positives together at the top with high probabilities. Among the three classifiers, XGBoost is the best classifier since it gives the steepest curve and the best AUC scores.

For AUC scores, all three classifiers achieve considerable results, with the highest of 0.91 from XGBoost. It means XGBoost has the probability of 91% to rank a randomly chosen positive sample higher than a randomly chosen negative sample.

5.4.2 Top-Recall

To get a better understanding of the classification capabilities of the classifiers, this section focuses on their performances on the top 20% accounts. The mean top-recall scores from the nested CV and the total training time are summarised in Table 5.1; the bold entry stands for the best score.

Table 5.1: Top recall scores and the training time of the five classifier. The training time is measured in seconds. Dsc_Tree stands for Decision Tree; Rand_Frst is Random Forest; XGB is XGBoost; Logi_Rgs is Logistic Regression; SVM is Support Vector Machine.

	Logi_Rgs	SVM	Dcs_Tree	Rand_Frst	XGB
Top-recall	0.40	0.39	0.64	0.79	**0.83**
Training time (s)	187	76	67	9138	55,748

The table shows that XGBoost achieves the best top-recall score: 0.83. It means that XGBoost classifies 83% of all genuine suspicious accounts at the top 20%, though it is also the most complicated model that needs the longest time to train. The other tree-based classifiers also achieve considerable results, and both of them are better than the linear models. The conclusion that tree-based classifiers are better than linear models and XGBoost is the best is consistent with the conclusion drawn from ROC results.

To see what features influence XGBoost most, Figure 5.5 shows the feature importance score for each feature. The importance score of each feature indicates its relative contribution to the model. The contribution is measured by the frequency of each feature that appeared in the trees in the XGBoost ensemble model. We can see that *outdegree* and *outinterval* have the most contributions and the 6 most "important" features are *In/Out-Degree*, *In/Out-NumAcct* and *In/Out-Interval*.

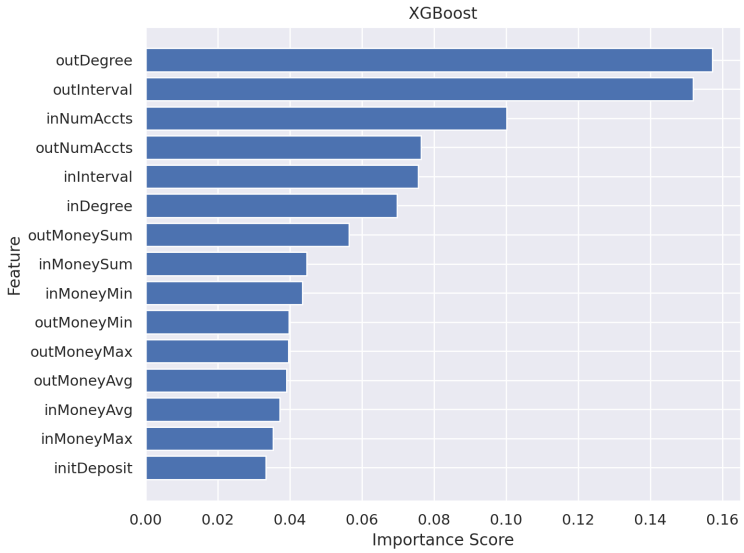


Figure 5.5: Feature importance scores for XGBoost

Degrees and the numbers of accounts connected directly reflect the spatial graph structures, while intervals reflect the temporal structures. The importance ranks then imply that graph structures play important roles in deciding whether an account is suspicious or not, and XGBoost is able to learn from graph structures. By contrast, all the features related to transaction statistics have lower ranks and much lower importance scores than those related to graph structures. It indicates that this type of features does not have significant impacts on classifiers. A plausible reason might be that although the money amounts in suspicious transactions are delicately designed to be realistic, the number of them is overwhelmed by the number of normal transactions. The dataset has totally 10,299,175 transactions yet only 974 suspicious transactions, the ratio is 9.5×10^{-5}. Since the money amounts in normal transactions are generated randomly (following a pre-defined distribution), the dataset can be seen that the money amounts in all transactions are generated randomly regardless of the labels of involving accounts. Therefore, the money amounts do not have a big impact on the classification decision of the classifiers.

Taking one step further, Figure 5.6 depicts the histogram of the number of accounts of each class at each probability predicted by Decision Tree. Comparing to XGBoost and Random Forest, Decision Tree outputs 36 probabilities, which is relatively a small number and can be depicted clearly. As the y axis is in log scale, if a count is 0, it will be removed.

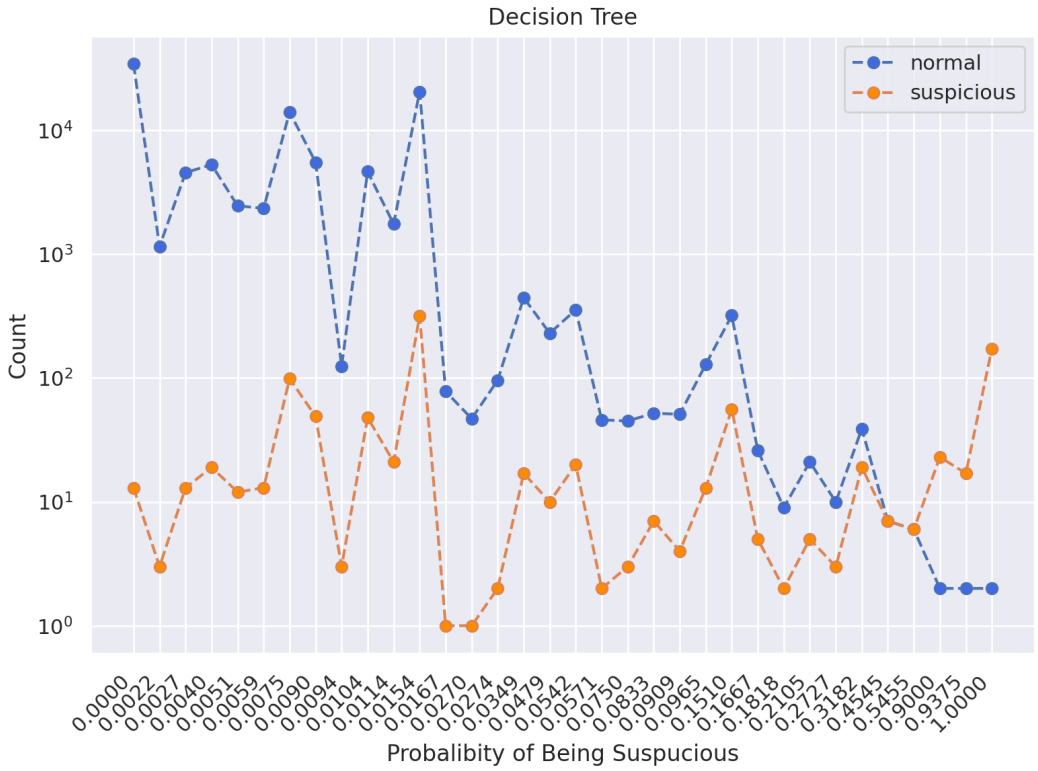


Figure 5.6: The number of normal and suspicious accounts at each probability that is predicted by a Decision Tree classifier.

First, we look at the orange points, which represent suspicious accounts. As the probability of being suspicious increases, the number of suspicious accounts at each probability does not increase globally as expected. But locally, at the rightmost part of the figure, from 0.27 to 1.00, there is a noteworthy increase tendency. This tendency shows that Decision Tree is more of a "conservative" classifier; it predicts high probabilities for those accounts it is confident about. This discovery also aligns with the steep part in the ROC curves. For the blue points, the number of normal accounts has a prominent decreasing tendency across the entire probability range. This indicates that the classifier can learn the patterns behind the normal accounts to a certain extent. Because it successfully excludes normal accounts from being assigned with high probabilities.

5.5 Summary

This chapter uses the nested CV strategy to evaluate the classification capabilities of five classifiers, namely Support Vector Machine, Logistic Regression, Decision Tree, Random Forest and XGBoost. Given 15 features extracted from the AML dataset for every account, each classifier tries to separate the accounts of the two classes from each other.

Their performance is measured by two measurements: ROC curves and AUC scores, and top-recall scores. Experimental results show that Linear Support Vector Machine and Logistic Regression have similar performance, outperforming a random guesser to a slight extent. The other three tree-based classifiers, by contrast, successfully learn from the features and separate the two classes. All three classifiers produce promising results when the classification threshold is high, meaning they are capable of correctly classify the most suspicious accounts. Among the successful classifiers, XGBoost yields the best performances across both measurements.

From the further investigation on XGBoost, the output feature importance scores show that features that reflect graph structures, both spatial and temporal, are more important than those that reflect the transactions statistics.

Overall, XGBoost will then be considered the representative classical classifier that compares with Graph Convolutional Network methods.

Chapter 6

Graph Convolutional Networks

In this chapter, we discuss the application of Graph Convolutional Networks (GCNs) to the AML problem. In Section 6.1 we define the problem that this chapter will address. Section 6.2 is the methodology section where we describe three key aspects: GCN fundamentals (Section 6.2.1), GCN variants (Section 6.2.2) and node embedding features (Section 6.2.3). In Section 6.3, we describe the experimental setup in detail, and in Section 6.4 we exhibit the results GCNs produce. Finally, in Section 6.5, we also re-train the best classical classifier XGBoost under the same settings to fairly compare it with GCNs.

6.1 Problem Definition

Following the problem definition in the previous chapter (Section 5.1), this chapter also treats the AML problem as a binary classification problem, where we use GCNs as classifiers. Similar to classical classifiers, GCNs are configured to output two probabilities, one for being suspicious and one for being normal, for every account in the dataset.

The question this chapter is going to answer is: comparing to classical classifiers, how is GCNs' classification capability? What are the influencers of their performance?

6.2 Methodology

This section presents two essential components for addressing the question shown in Seciton 6.1. In Section 6.2.1, we describe GCNs in detail. In Section 6.2.2, we show three GCN variants that will be investigated in the experiments. In Section 6.2.3, we describe a graph embedding method called *node2vec*, whose output will be used to input to GCNs.

6.2.1 GCN Fundamentals

The data for typical deep learning applications such as image classification (Rawat and Wang, 2017) is represented in a table, where each record (row) is an image, and a number of pixel values (columns) are its features, one extra column shows the label of the image, which is expected to be predicted by deep neural networks. Records are expected to be independent and identically distributed in such applications. In AML, however, there are relations between different data records; a simple table is not sufficient to represent AML datasets. GCNs are the tools to address the problems involving such type of data.

The label prediction of a data point made by GCNs depends on not only the features of that single data point but also the data points with which it has relations. GCNs learn from those features and update their kernel weights under the supervision of known labels in iterations. Once the updating process (training) is done, GCNs are used to predict the missing labels of the accounts in the graph.

A GCN model learns a parameterised function that takes a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as input and produces a feature matrix $\mathcal{H} = |\mathcal{V}| \times D_f$ which consists of feature vectors for all vertices (or nodes) in the graph. To clarify the notations above, $\mathcal{V}$ represents all vertices in the graph and $\mathcal{E}$ the edges, $|\mathcal{V}|$ and $|\mathcal{E}|$ are their numbers; D_f, the length of output feature vectors, is customisable and depends on the task GCNs are used for. In the AML problem, a (transaction) graph is constructed by representing accounts as vertices and transactions as edges. If there are multiple transactions between the same pair of accounts, all the transactions will be presented in the graph. So, the transaction graph is a directed multigraph.

Besides the graph structure information that is immersed in the graph, nodes and edges may also convey information by means of node feature representations $\mathcal{P} = |\mathcal{V}| \times D_v$ and edge feature representations $\mathcal{Q} = |\mathcal{E}| \times D_e$; D_v is the length of node features and D_e is that of edges. The node and edge features are computed from the data; their lengths are thus depending on how many features are extracted from the data. For example, the total number of transactions an account made can be a node feature for the account; the amount of money in a transaction can be an edge feature for the transaction. These features are the messages of the nodes and the edges in the graph. These two feature representations,

along with the graph structure, form the input for GCNs.

Figure 6.1 shows the pipeline for applying a GCN on a transaction graph (edge features are ignored for simplicity). A input graph goes through a GCN with multiple GCN layers, and the GCN outputs the probabilities of being suspicious and of being normal for every node in the graph. Each GCN layer updates the node feature for every node; the updating process can be summarised into three steps: *message passing*, *message aggregation* and *graph convolution.*

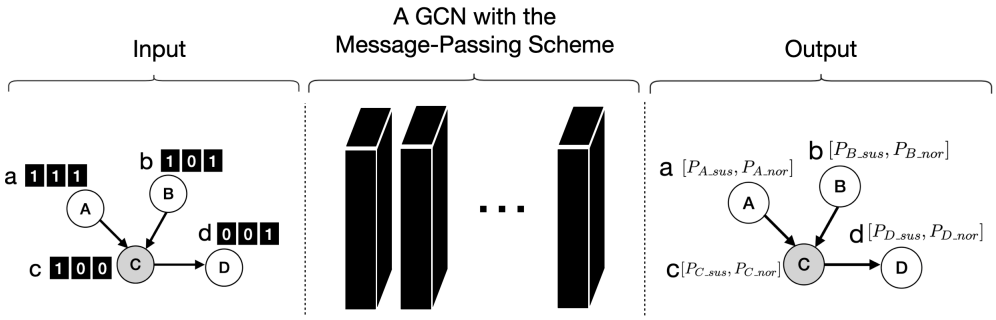


Figure 6.1: A GCN pipeline for the AML problem, which inputs a graph with node features and edge features (if applicable) to a GCN, and outputs two probabilities of being suspicious or normal for every node. Every layer in the GCN updates the node features; the final features are the probabilities.

Message Passing

Message passing enables GCNs to explore the graph structure of an input graph. Every node in the graph receives *messages* from its incoming neighbours. A message can be the node feature of a neighbour or a combination of the node feature and the edge feature of the edge that connects the two nodes. What a node receives may differ from what was sent because different *message passing functions* may apply to the messages. For example, if the feature of a neighbour node was sent together with the edge feature, the received message could be the product of these two features or a result of any other binary operation. Even without edge features, binary operations can also be applied to the features of the neighbour nodes and that of the receiving node itself. Yet these message mutations are not mandatory; nodes can receive from their neighbours their original node features.

Taking the graph in Figure 6.2 as an example, the graph consists of four nodes and their corresponding features as well as three directed edges. When applying message passing on node C, it receives the messages (node features) from node A and B since they are its incoming neighbours. Node D is node C's outgoing neighbour thus C does not receive its

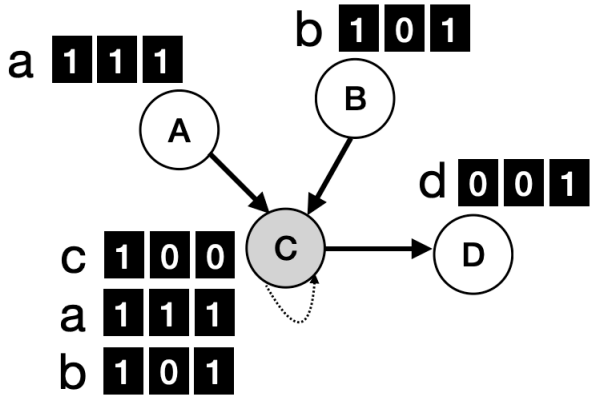


Figure 6.2: An example graph for illustrating message passing.

message. In such a way, nodes only gather messages from their incoming neighbours but not themselves. However, in identifying whether a node is suspicious of money laundering or not, the node's own feature is the primary interest. Therefore, a self-loop edge is added for every node in the graph to make the nodes able to receive their own messages. For a node $v \in \mathcal{V}$, message passing can then be formulated as the following:

$$\mathcal{M}_v = \Psi(p_u, q_{u\_v}, p_v), \forall u \in \mathcal{V} \quad s.t. \quad e_{u\_v} \in \mathcal{E} \tag{6.1}$$

where $\mathcal{M}_v$ denotes all messages node v receives; Ψ is a message passing function that takes the node features of both v (p_v) and of its incoming neighbours (p_u) as well as the edge feature (q_{u_v}) of the edge that connects the two nodes. For a message passing function, it is not necessary to have all three components presented. Note that in Equation 6.1, after adding self-loop edge for every node ($e_{v\_v} \in \mathcal{E}$), node v also comply the restraints for u; thus $\mathcal{M}_v$ includes the message from node v itself.

To summarise, the message passing step, for every node in the graph, collects messages from incoming neighbours as well as themselves. Message passing only occurs between neighbours thus can be seen as exploitation on the graph structure.

Message Aggregation

This step is called message aggregations or message summarisation or message reduce. Every node, in this step, aggregates the (possibly mutated) messages from its neighbours as well as itself. All received messages are aggregated into one feature by *message aggregation*

functions such as summation, mean. For a node $v \in \mathcal{V}$, message aggregation can be formulated as the following:

$$s_v = \Phi(m_v^1, m_v^2, ..., m_v^{|\mathcal{M}_v|}) \tag{6.2}$$

where s_v is the aggregated feature of node v; Φ is a message aggregation function; m_v^i is a message arrived at node v from its i-th incoming neighbour, which is part of $\mathcal{M}_v$. The aggregated message of a node is its temporary representation, which is the input of the graph convolution step.

Graph Convolution

The last step applies convolution calculations on the summarised features. For a node $v \in \mathcal{V}$, the following equation illustrates such a calculation:

$$p'_v = \sigma(s_v \cdot \mathcal{W}) \tag{6.3}$$

where s_v is the aggregated feature of node v; $\mathcal{W}$ a learnable parameterised kernel (weight matrix), which is also shared among all convolution calculations for all nodes in the same convolutional layer; σ is a non-linear activation function such as ReLU; p'_v is the output of the graph convolutional layer and is the new feature representation for node v, which will be taken as the input for the next layer; or if it is in the last layer of a GCN, p'_v will be the final output of the GCN of node v.

The size of the convolutional kernel in each GCN layer is partially customisable. Its row dimension is dependent on the column dimension of s_v yet its own column dimension can be customised arbitrarily. Special attention should be given to the kernel design of the last (output) layer since the column dimension of which determines the dimension of the final outputs. This chapter uses GCNs as binary classifiers; thus, the desired output dimension is 2. *Softmax* activation function is then applied to the GCNs output, producing the probability of being in the suspicious class and that of being in the normal class for every node in the graph respectively.

The goal of training a GCN is to make the GCN learn and find the optimal weights in all the kernels by iteratively updating them. The updating process is supervised by a loss function calculated on the predictions the GCN produces and their corresponding true labels. In this specific binary classification problem, the *cross-entropy* between the predictions and the true labels is used as the loss function:

$$loss(\mathbf{y}', \mathbf{y}) = -\sum_i \ \mathbf{y}_i \ log(\mathbf{y}'_i) \tag{6.4}$$

where $\mathbf{y}'$ is a prediction vector with a length equals to the number of total classes (corresponds to the column dimension of the kernel in the last GCN layer), the value at each index is the probability of being to the class the index represents; $\mathbf{y}$ is a vector with the same length as $\mathbf{y}'$, which has a value of 1 at the index representing the true class and 0's at other indices; $\mathbf{y}'_i$ is the value at the i-th index of the prediction vector.

The value of the cross-entropy loss function becomes zero when the prediction is perfect (the probability of being in the correct class reaches 1.00 and 0.00 for other classes). It gives minor penalties when the prediction and the true labels are close to each other and produces exponential penalties otherwise. In such way, it forces classifiers to separate different classes so that the penalties would be low. The Stochastic Gradient Descent (SGD) algorithm (Ruder, 2016) is used to minimise the loss function (penalty).

Once the training is done, e.g. the loss becomes lower than a pre-defined threshold, a GCN with its learnt kernels can be used to predict the labels of new nodes. Given a new node and its graph context, the GCN applies message passing, message summarisation, graph convolution steps on it and outputs the probabilities of being either account type.

Summary

A GCN takes a graph as input; specifically, graph structures and node/edge features. Node/edge features are presented in the form of vectors, and graph structures can be presented as adjacency matrices, for example. In each layer in the network, every node receives messages from its incoming neighbours and summarises all messages, including its own, into one single intermediate feature vector. A shared convolutional kernel in each layer then applies to the intermediate feature vector, producing a new feature representation for the nodes. The output of GCNs is transformed into probabilities with the help of Softmax activation. The kernels in GCN models are trained under the supervision of the cross-entropy loss function; the training ends when the loss value is lower than a pre-defined threshold or when it reaches the training time limit. The final trained GCN models are then used to predict the labels of the nodes other than those in the training set.

6.2.2 GCN Variants

Different GCN variants exploit graph information in different ways by applying different messaging passing functions and different message aggregation functions. To obtain a more

comprehensive understanding about the effectiveness of GCNs, three GCN variants will be investigated. This section describes the variants in detail.

GCN-Vanilla GCN-Vanilla (Kipf and Welling, 2016) networks stack multiple GCN layers sequentially. Layers in GCN-Vanilla use *copy source* as the message passing function, which purely copies the intact node feature of the source nodes to the message receiving nodes. The message aggregation function used is *degree normalisation*, which is defined as follows for a node $v \in \mathcal{V}$:

$$s_v = \frac{1}{\sqrt{|N(u)|}\sqrt{|N(v)|}} \sum_{u \in N(v)} m_v^u \tag{6.5}$$

where m_v^u is the message node v received from its in-coming neighbour u, $N(u)$ is the in-coming neighbourhood of node u and thus $|N(u)|$ is its in-degree.

In GCN-Vanilla, the output of the previous layer is directly fed into the next layer until the final output layer produces the probability vector for every node.

GCN with Edge Features (GCN-EF) GCN-EF integrates edge weights into its GCN layers. The only difference between GCN-EF and and GCN-Vanilla is their message passing function. The one GCN-EF uses is *source_multipy_edge*, which is defined as follows for a node $v \in \mathcal{V}$:

$$\mathcal{M}_v = p_u \odot q_{u\_v}, \quad \forall u \in \mathcal{V} \quad s.t. \quad e_{u\_v} \in \mathcal{E} \tag{6.6}$$

where u represents all the incoming neighbours and p_u the node features; $\odot$ is the element-wise multiplication operator. For the multiplication, if the lengths of the node feature and the edge feature are not the same, the shorter one will broadcast its feature to a new length to be the same as the longer one. In this AML case, the edge feature is the amount of money transferred in the transaction, thus a simple scalar. Therefore, the *source_multiply_edge* function produces features with the same lengths as of original node features, with the values in which scaled by the amount of money.

Graph Attention Network (GAT) GAT uses a more complicate network structure comparing with the previous two variants. It introduces an attention layer (Bahdanau et al., 2014) into each GCN layer, for a node $v \in \mathcal{V}$, the output of a GCN layer becomes:

$$p'_v = \sigma(\sum_{u \in N(v)} \alpha_{u,v} \cdot s_v \cdot \mathcal{W}) \tag{6.7}$$

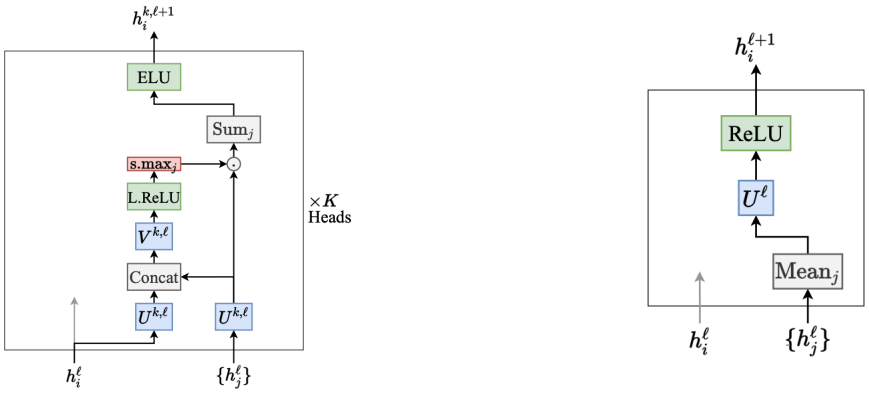
where $\alpha_{u,v}$ is the attention score between node u and v:

$$\alpha_{u,v} = \text{softmax}(l_{u,v}) \tag{6.8}$$

$$l_{u,v} = \text{LeakyReLU}(\boldsymbol{a} \cdot (p_v \cdot \mathcal{W} \;||\; p_u \cdot \mathcal{W})) \tag{6.9}$$

where LeakyReLU is an activation function that allows a small gradient for inactive units: $\text{LeakyReLU(x)} = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$; $\mathcal{W}$ is the layer kernel; $\boldsymbol{a}$ is a learnable attention vector.

Figure 6.3a visualises a GAT layer; Figure 6.3b shows a GCN layer. GATs is significantly more complicated comparing with GCN-Vanilla. GAT also applies a multi-head scheme to improve its learning capacity by averaging the output of the multiple heads.



(a) GAT

(b) GCN-Vanilla

Figure 6.3: Graph structures of GAT and GCN; adopted from (Dwivedi et al., 2020).

6.2.3 Embedding Features

Graph embedding methods (Goyal and Ferrara, 2018) take a graph as the input and produce a vector representation for all nodes in the graph. Such representations can be used as node features, and to be the input of GCNs. The graph embedding method *node2vec* (Grover and Leskovec, 2016) maps the nodes in a graph to vector representations, which is in the form of a matrix: $\mathcal{Z} = |\mathcal{V}| \times d$, where d is the dimension of each feature representation. To learn the weights in the matrix, node2vec maximises the log-probability of a neighbourhood

$N_S(v)$ for a node v; $N_S(v)$ is neighbourhood sampled according to a strategy S. Formally, the learning objective function is defined as the following:

$$\max_{\mathcal{Z}} \sum_{v \in \mathcal{V}} [-\log R_v + \sum_{n \in N(v)} \mathcal{Z}(n) \cdot \mathcal{Z}(v)] \tag{6.10}$$

where $R_v = \sum_{u \in \mathcal{V}} \exp(\mathcal{Z}(v) \cdot \mathcal{Z}(u))$ is a per-node partition function. Equation 6.10 is then maximised by stochastic gradient ascent (SGD) to produce the optimal feature representation matrix $\mathcal{Z}$.

The major innovation in node2vec is its neighbourhood sampling strategy S. For every node v, a number of 2^{nd} order random walks (k) with a certain length (l) are performed; the resulting walk trajectories are the neighbourhood $N_S(v)$ used above. The 2^{nd} order random walks are guided by two parameters p and q. Let a random walk that has reached to node v from node t and now is selecting its next walk destination x, the transition probabilities $\pi_{v,x}$ is defined as $\pi_{v,x} = \alpha_{p,q}(t,x) \cdot w_{v,x}$, where

$$\alpha_{p,q}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases} \tag{6.11}$$

and $d_{t,x}$ is the length of the shortest path between node t and v; $w_{v,x}$ is the edge weight of $e_{v,x}$. Figure 6.4 shows an example of such a walk.

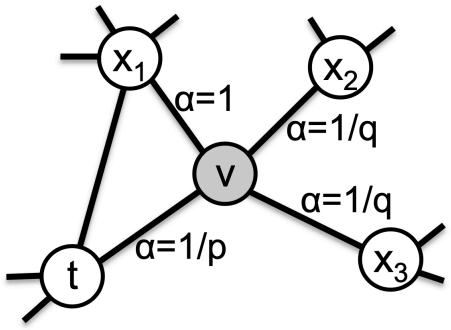


Figure 6.4: An example of 2^{nd} random walk process; adopted from (Grover and Leskovec, 2016).

To summarise, node2vec performs a number of random walks starting at every node in the graph, and the resulting walk trajectories are then considered to be neighbourhoods. It

defines an objective function that maximises the log-probability of the neighbourhoods, the result of which is an optimised feature representation matrix. The matrix that represents each node in the graph with a feature vector is the final output of node2vec.

The way node2vec samples the neighbourhoods explores the graph structures and integrates them into the output. Performing multiple random walks for each node reduces the variance. Both of the characteristics provide promising feature representations for the AML problem since detecting suspicious money laundering pattern lies in exploring the local graph structures in transaction graphs.

6.3 Experimental Setup

This section describes in detail the experimental setup: how GCNs are used as binary classifiers. It starts with presenting the first part of the dataset: graph data, and showing its special usage in the AML problem in Section 6.3.1. As another essential part of the dataset, different types of node features are individually explained in Section 6.3.2. And finally Section 6.3.3 shows the hyper-parameter tuning strategy.

6.3.1 Graph Data

The graph dataset r100K that contains 100,000 nodes described in Chapter 4 is used for all experiments in this chapter. Unlike in Chapter 5 where the dataset is used in a nested cross-validation manner, the dataset is now split into three sets (training/validation/test) for GCN experiments. This means that all experiments are trained on the same training and validation set and are tested against the same test set.

The reasons why nested cross-validation is not applied to GCNs are: (1)GCNs are computationally expensive. Besides that, the run time of a single epoch is much longer than that of classical methods; the number of training epochs, the number of hyper-parameter searches already drastically increase the run time of a single experiment. Adding cross-validation on top of that would make the run time unrealistic. (2) The purpose of GCN experiments is no longer only to verify its classification capability on the AML dataset. Instead, it is to build a working model, which can be further thoroughly compared with the best classical methods. For this purpose, cross-validation is not suitable since it would produce k different models, in the case of k-fold cross-validation. The nature of cross-validation itself is not designed for model building but for model evaluation.

The dataset is split according to the following method. First, it is divided into five splits using the *stratified k-fold* splitting strategy such that each split preserves the same ratio of the number of suspicious accounts over the number of normal accounts. Among the five

splits, one split is reserved as the test set, one is the validation set and the rest three form the training set. Thus 80% of the dataset is used for training (60% comes from the training set and 20% the validation set), 20% of the dataset is taken as the unseen test set. This static splitting of training/validation/test sets will be consistent across all experiments in this chapter.

The result of the dataset splitting is that the index of every node in the graph is assigned to one of the three sets. For example, if the account that has an index of 1 is assigned to the training set, then it will be a part of the training set throughout all experiments regardless of which type of node feature is attached to it in different experiments.

Despite the dataset splitting, it does not mean GCNs train on the subgraph that only contains the nodes that belong to the training set; this would entirely ruin the graph structure and undermine the information the graph conveys. Instead, the training is conducted on the entire graph, and the nodes in the training set are able to receive messages from their neighbours even they are not part of the training set. The kernel weight updating process only depends on the loss calculated on the nodes belonging to the training set. In other words, although the nodes outside the training set are reachable, their labels are not. Only the labels of the nodes in the training set are seen during the training, and the training is only supervised by them. The validation and test processes are similar: the entire graph is used for message passing, but only the performances on the corresponding nodes are reported.

6.3.2 Node Features

The experiments use two types of node features. The first type presents the properties of accounts and transactions, which is human-readable and has been used in classical classifiers in Chapter 5. The second type is the output of the graph embedding method *node2vec*; this type of features is not human readable.

The property-related features and the graph embeddings produced by node2vec are the two feature types that are used in this chapter. The eight node feature sets are summarised as follows.

1. Property. The properties extracted from accounts and transactions such as in-degrees, the amount of money transferred. This feature set is the same as the one used in classical classifiers, and as described in Section 5.3.1. The name of this feature set will be simplified to "X".

2. Property-Standardised. A standardisation version of *Property*. That is, each feature in *Property* is normalised according to $X' = \frac{X-\mu}{\sigma}$, where X' is the normalised feature, X is the original feature in *Property*, μ and σ being the mean and the standard deviation of feature X. The name of this feature set will be simplified to "X_std".

3–8 n2v-1 - n2v-6. Node features (embeddings) produced by six node2vec (n2v) models with different hyper-parameters. The specific settings are summarised in Appendix Table A.6.

Each of the eight feature sets will be individually attached to the nodes in the transaction graph to train a model. Therefore, for every GCN variants, there will be eight models to be trained on the transaction graph with the eight different node feature sets attached.

6.3.3 Hyper-parameter Tuning

Hyperopt (Bergstra et al., 2013) is used for automatic hyper-parameter tuning in the experiments. Given a hyper-parameter search space and an objective function, hyperopt is able to explore the search space and narrow it down to the best hyper-parameter set that minimises (or maximises) the objective function.

In Section 6.2.1, we discussed GCN fundamentals, where we can find a couple tuneable hyper-parameters in GCNs. Besides, there are three tuneable hyper-parameters regarding the training process.

1. The depth of a GCN. As shown in Figure 6.1, a GCN consists of multiple layers; the number of layers can be set to any number. In the following experiments, the depth of a GCN refers to the number of GCN layers in it, excluding the input and the output layer.
2. The kernal size of a GCN layer. As shown in Equation 6.3 shows, the graph convolution step in a GCN layer applies a convolutional kernel to the node features. The size of the kernel is customisable.
3. The dropout rate in a GCN layer. It is the probability with which an element of the input of the layer to be zeroed. Applying dropout could prevent models from overfitting (Srivastava et al., 2014).
4. The weight decay factor of the training. It is the penalty factor applying to model weight; it also serves the purpose of preventing overfitting.
5. The learning rate of the training. It defines the step size at each iteration of minimising a loss function.

The above 5 hyper-parameters will be tuned for GCN-Vanilla and GCN-EF; the search space is summarised in Appendix Table A.7. For GAT, because of its complexity, hyper-parameter tuning is not conducted on it.

In the experiment for GCN-Vanilla and GCN-EF, hyperopt conducts 100 searches, each search delivers arbitrarily 1000 epochs, each epoch is evaluated on the validation set using

the top-recall metric. The objective function for *hyperopt* to maximise is the maximum top-recall score of each search. That is, each search is represented by the maximum top-recall achieved by any epoch within the search; hyperopt tries to find the search that maximises those representatives. The reason for not using the top-recall from the last epoch is, on the one hand, that models can be overfitted after being trained in a large number of epochs. On the other hand, the maximum is a reasonable representation of a search regardless in which epoch the performance is achieved; since it shows that the specific maximum performance is achievable under such a hyper-parameter setting.

6.3.4 Implementation Information

Their implementations of the three GCN variants are based on Deep Graph Library (Wang et al., 2019), which uses PyTorch (Paszke et al., 2019) as the backend.

The experiments for GCN-Vanilla and GCN-EF are conducted on the GPU GeForce GTX 980 Ti with 6 GB memory. The experiments for the more complex network GAT are conducted on the GPU GeForce GTX Titan X with 12 GB memory.

6.4 Experimental Results

This section presents the experimental results of GCN variants: GCN-Vanilla (Section 6.4.1), GCN-EF (Section 6.4.2) and GAT (Section 6.4.3) respectively.

6.4.1 GCN-Vanilla

Best Search

The eight sets of node feature mentioned in Section 6.3.2 are applied to GCNs. Table 6.1 summarises the performance of the models trained on the two property feature sets and that on n2v-5, which is the best version of the node2vec models. Table 6.2 summarises the rest versions of the node2vec models. The models are not only evaluated on the test set but also on the full dataset. This is because of the speciality of graph learning: nodes in different sets are not entirely isolated. Both in training and testing, nodes in other sets are still reachable. Therefore, considering all nodes in the graph as one and reporting the model performance on the full set provide a supplementary reference for model performances.

Table 6.1: Performances of GCN with different input features.

Evaluation Set	Test Set			Full Set		
Node Feature	X	X_std	n2v-5	X	X_std	n2v-5
Top-recall	0.23	0.22	**0.29**	0.24	0.22	**0.29**
ROC-AUC	0.55	0.48	**0.58**	0.53	0.50	**0.57**

Table 6.2: Performances of GCN with different node2vec input features.

Evaluation Set	Test Set					Full Set				
Node Feature	n2v-1	n2v-2	n2v-3	n2v-4	n2v-6	n2v-1	n2v-2	n2v-3	n2v-4	n2v-6
Top-recall	0.24	0.27	0.28	0.23	0.20	0.23	0.24	0.24	0.22	0.23
ROC-AUC	0.55	0.53	0.57	0.51	0.49	0.54	0.52	0.54	0.50	0.51

We can see in Table 6.1 that model n2v-5 outperforms both property models in terms of both top-recall and ROC-AUC. For top-recalls, model n2v-5 achieves at least 6% higher than property models on the test set. However, looking at both tables, even the best model among all eight models is not able to achieve a top-recall score higher than 0.30. And all

the models have ROC-AUC scores lower than 0.60, which indicates that most models only perform slightly better than random guessing.

Since every model has a top-recall between 0.2 and 0.3, meaning 20% to 30% genuine suspicious accounts are covered by the top accounts. This section also explores whether the suspicious accounts that different models correctly predicted are similar, for example, whether the suspicious account all models correctly predict are all indexed by from 600 to 800 and they fail to correctly predict other suspicious accounts.

Jaccard Similarity (Jaccard, 1912) is used to evaluate the similarity between model predictions. For each model, the indices of the true positive (TP) accounts form a TP set. The Jaccard Similarity between two models is then calculated as the size of the intersection of their TP sets divided by the size of the union of them, which can be formulated as follows:

$$J(M_a, M_b) = \frac{|TP_{M_a} \cap TP_{M_b}|}{|TP_{M_a} \cup TP_{M_b}|} \tag{6.12}$$

First, we look at the results from "X" and "X_std" since they belong to the same type of node features. The results are summarised in Table 6.3. The two models correctly predict 238 and 217 suspicious accounts respectively. These numbers are rather low; however, the number of suspicious accounts they both correctly predicted (intersection) is only 65, while the union of the two TP sets has the size of 390. This shows that the two models predict different groups of suspicious accounts.

Table 6.3: Similarity between the predictions of model X and model X_std.

	X	X_std
Num Correct Predictions	238	217
Intersection Size	65	
Union Size	390	
Jaccard Similarity	0.17	

The same situation also happens in the six node2vec models. In this case, the average Jaccard Similarity is used since Jaccard Similarity is only suitable for two-set comparison. In multi-set scenarios, the sizes of the intersection could be small, and the sizes of the union could be large as the number of sets increase, the quotient of them would be difficult to interpret. Therefore, among the six models, $\binom{6}{2}$ intersection and union are calculated and then are averaged. Table 6.4 summarises the results. Averagely, the results are similar to that of the property models: every pair of models has a small intersection size and relatively a large union size. This could be an indication that an ensemble model that combines the models that are trained on different node features can give better predictions.

Since different models produce non-similar predictions, an ensemble model can collect their "expertise" at different areas, thus correctly predict more genuine suspicious at the top.

Table 6.4: Similarity between the predictions of the six different node2vec models

	n2v-1	n2v-2	n2v-3	n2v-4	n2v-5	n2v-6
Num Correct Predictions	229	239	244	226	294	228
Avg. Intersection Size			86.67			
Avg. Union Size			400.00			
Avg. Jaccard Similarity			0.22			

Another assumption is that the non-similarity in the predictions is caused by the incapabilities of the models. Their predictions are similar to random guesses. Top-recall scores are calculated under the condition of assuming the top 20% accounts are predicted as suspicious by models, thus a random guesser could also rank 20% of genuine suspicious accounts at the top 20% of all accounts and achieves 0.2 top-recall, which is similar to what the models achieved.

Key Influencers

With the purpose of improving the performance of the models discussed above, this subsection investigates which aspects of GCNs are the key influencers of their performance.

All experiments in this part have a base hyper-parameter setting, which is shown in Appendix Table A.8. When investigating the influence of the individual aspects, only the corresponding hyper-parameter will be changed. All experiments are performed on the n2v-5 node feature set since it yields the best performance in the previous experiments. All results reported of the following experiments are averages over five identical experiment runs.

Network Depth In each GCN layer, Massage Passing allows a node to receive messages from its direct (1-hop) incoming neighbours, and these messages will later be transformed into the node's new feature. The new feature will be passed to its outgoing neighbours in the next layer. Therefore, in the i-th layer of a GCN network, a node is able to integrate features from its i-hop incoming neighbours. In other words, the depth of a GCN network determines how far a node is able to reach its neighbours. This can be taken into account when designing GCN networks. AML problem only concerns local structures; thus, a shallow network might be more efficient.

Figure 6.5 shows the performance of GCN networks with different depths. It plots the top-recall scores on the validation set throughout the training process. In the legend, "val set" tells the maximum top-recall achieved on the validation set during the training, "test set" and "full set" also show the best performances. Note that the depth means the number of hidden layers in a GCN network. For example, with a depth of 0, a vanilla GCN network has only an input layer and an output layer.

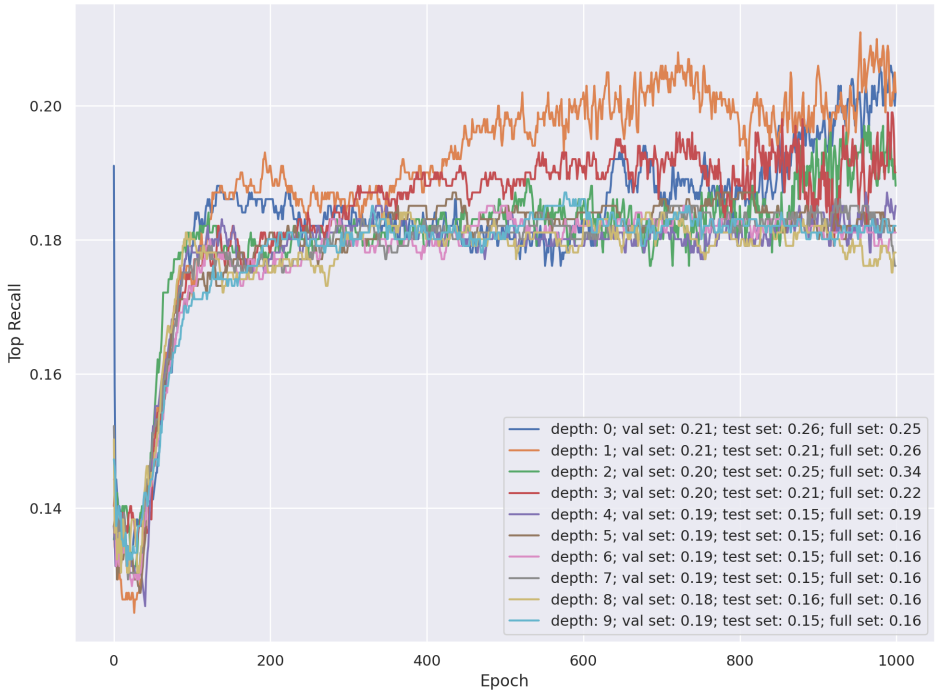


Figure 6.5: The Performance of Vanilla GCNs with different depths.

From Figure 6.5 we can see that with depths deeper than 3, the performances of GCNs decrease dramatically. While with depths lower than 3, vanilla GCNs maintain a relatively high level of performance and achieves the best results, in terms of "full set", at the depth of 2. The results support the assumption that shallow networks are more capable of capturing AML patterns that exist on local structures. And thus they achieve better performance.

Kernel Size The dimension of output node features of a GCN layer is defined by the number of hidden units (kernel size) the layer contains. In GCNs, every hidden layer has the same number of hidden units while the output layer has another arbitrary number of hidden units; in this AML problem, it is 2.

Figure 6.6 summaries the performances of GCNs with the kernel sizes range from 2^1 to 2^{10}. Their performances on the validation set do not vary very much: top-recall scores are all around 0.2. However, on the full set, their performances almost monotonically increase as the kernel sizes increase. The model with 2^{10} hidden units achieves 0.66 top-recall on the full dataset, which is a considerable increase as all results in Section 6.4.1 are around 0.2.

The reason why previous results never reach the performance of this level is that the search space for *hyperopt* does not consider such a large kernel size due to the consideration of limiting the search space. If the kernel size is set to choose from the 10 values listed in the figure uniformly, the entire search space would increase tremendously. However, the number of searches is pre-defined, complying with the limited computational resource. Therefore, the search space has to be concrete otherwise it would be diluted, and the tuning process would fail to produce a solid result.

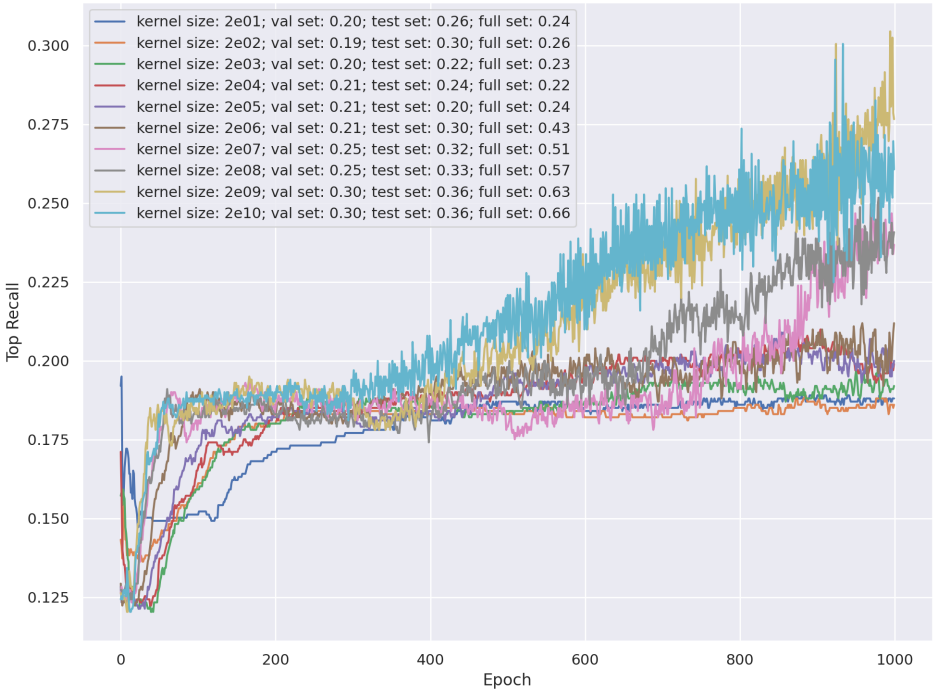


Figure 6.6: The Performances of GCN networks with different kernel sizes

However, despite the encouraging results on the full dataset, results on the validation set and the test set are still not satisfying. This situation leads us to the infamous problem that models perform well on the training set yet poorly on the test set.

Digging into the model with a kernel size of 2^{10} that achieves 0.66 top-recall, it correctly predicted 663 suspicious accounts (663/1007≈0.66), which covers 532 out of 605 (≈88%) suspicious accounts in the training set, 66 out of 201 (≈ 33%) in the validation set, 53 out of 201 (≈ 32%) in the test set. This phenomenon is similar to a classical overfitting problem, where models achieve good performance on the training set by "memorising" it during the training process yet are not able to learn the patterns behind the data nor produce good performance on the test set. Despite this, achieving a top-recall score of 0.32 on the test is a 50% increase over the models in the previous subsection, which indicates that increasing the kernel size of a vanilla GCN network does have positive effects on the model performance.

6.4.2 GCN with Edge Feature (GCN-EF)

The amount of money a transaction transferred will be considered as the weight of its corresponding edge. Instead of the original amounts, the normalised amounts are used. They are normalised according to the rule: $X' = \frac{X-\mu}{\sigma}$, the same way "X_std" is standardised.

Figure 6.7 shows the distribution of the original raw amounts and that of the standardised amounts. The shapes of them are similar; the main difference lies in the x-axes: the range of the money amounts. The raw amounts range from 1 to 5M, which would extremely deviate the relatively concentrated node features when the amounts as edge features apply. As a result, the values of node features can be exploded and thus make the training difficult. By contrast, the normalised amounts range from about -1 to 10. The narrow range can be more feasible regarding avoiding feature exploration.

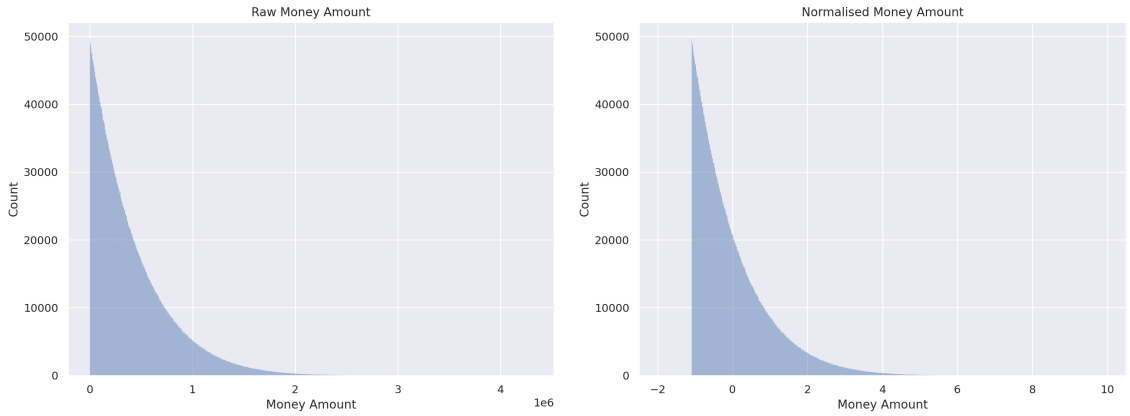


Figure 6.7: Distributions of raw money amount (left) and normalised money amount in all transactions (right).

With the same setting as in GCN-vanilla, this subsection also tried different network depths and kernel sizes. The results show that GCN-EF models are not able to achieve the high top-recall scores that are achieved by GCN-Vanilla with large kernel sizes. Comparing to the best top-recall of 0.66 on the full dataset from GCN-Vanilla, GCN-EF is only capable of producing the best top-recall of 0.28. Also, GCN-EF models are more unstable. Their performances fluctuate over all epochs, showing multiple crests and troughs in the curves comparing with their GCN-vanilla counterparts. The performances of GCN-EF with different depths and kernel sizes are shown in Appendix Figure A.1 and Figure A.2 respectively.

According to the experimental results, this sub-section concludes that adding edge features (money amount in transactions) to GCN makes models more unstable and not able to introduce more information to the models and improve them.

Plausible reasons are: (1) Despite that the money amount in suspicious transactions are delicately designed to be realistic, the number of them is too few. The money amounts in suspicious transactions are too overwhelmed to make a difference. (2) Most suspicious transactions involve amounts less than 10,000 (79% in Table 4.4), their normalised counterparts are at the left part of Figure 6.7 (the right sub-plot), whose values are relatively small; thus, they may not be able to affect the calculations on node features.

6.4.3 Graph Attention Network

Due to the high complexity of GAT that its running time for one epoch is about 40 times longer than that of GCN-vanilla; in this sub-section, we do not apply hyper-parameter searching for GAT. As we found that the size of kernels has a significant influence on the performance of GCN models, GAT models with different kernel sizes are trained and the results are summarised in Figure 6.8.

In contrast to the monotonic increase in the performances of GCN-vanilla, GAT stabilises when the kernel size reaches 2^4 and even starts to decrease when the kernel becomes larger than 2^7. But overall large kernels produces better performance.

The most prominent information this figure conveys is that GAT does not only achieves good performance on the full set but also on the validation set and the test set. Especially, the GAT model with a kernel size of 2^5 achieved **0.56** top-recall on the test set. This is a strong indication that GAT models learnt the patterns behind the AML data and are able to generalise its classification capability to unseen data.

To compare the classification capability of the three GCN variants, Table 6.5 summarises the performance of their best models.

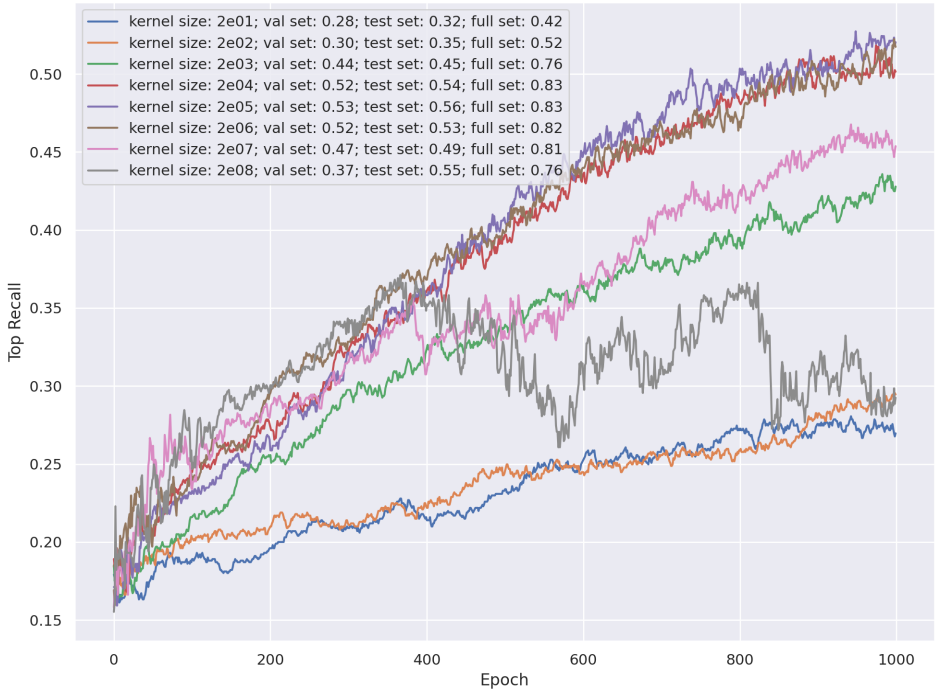


Figure 6.8: The performance of GAT with different kernel sizes on the validation set.

Table 6.5: Performance and training time of the best models of the three GCN variants.

Evaluation Set	Test Set		Full Set		Training Time
GCN Model	Top-recall	ROC-AUC	Top-recall	ROC-AUC	Per Epoch (s)
GCN-Vanilla	0.36	0.60	0.66	0.77	0.57
GCN-EF	0.26	0.52	0.28	0.54	0.62
GAT	**0.56**	**0.72**	**0.83**	**0.90**	1.73

Among the three GCN variants, GAT significantly outperforms the other two variants. Despite the training time per epoch of GAT is 1.26 seconds more than that of GCN-Vanilla, its 0.56 top-recall score on the test set is a 56% ($\frac{0.56-0.36}{0.36}$) improvement over GCN-Vanilla. The improvement shows that the performance of GCN models is sensitive to network complexity. It has shown that increasing the network complexity by increasing the kernel sizes in GCN-Vanilla improves the performance of the models. GAT, a more complicated network, produces further improvements.

6.5 Re-Train XGBoost

In Chapter 5, we concluded that XGBoost is the best classical classifier. To have a more objective comparison of the power of XGBoost with GCNs, we re-trains new XGBoost models using the same dataset split that is used by GCN models. Also, in the original settings, XGBoost only uses feature sets "X" and "X_std"; this section applies all the node feature sets used by GCNs on XGBoost.

Table 6.6 summarises the results of XGBoost models trained on property feature sets and the most successful node2vec version "n2v-5"; the rest of node2vec versions are summarised in Table 6.7.

Table 6.6: Performance of XGBoost trained on the two property feature sets and the most successful n2v feature set. Results are averages from three experiment runs with different random seeds.

Evaluation Set	Test Set			Full Set		
Account Feature	X	X_std	n2v-5	X	X_std	n2v-5
Top-recall	1.00	0.42	**1.00**	0.99	0.43	**1.00**
ROC-AUC	1.00	0.68	**1.00**	1.00	0.67	**1.00**

Table 6.7: Performance of XGBoost trained five n2v feature sets. Results are averages from three experiment runs with different random seeds.

Evaluation Set	Test Set					Full Set				
Node Feature	n2v-1	n2v-2	n2v-3	n2v-4	n2v-6	n2v-1	n2v-2	n2v-3	n2v-4	n2v-6
Top-recall	1.00	0.77	0.99	0.82	0.99	1.00	0.79	0.99	0.79	0.99
ROC-AUC	1.00	0.87	0.99	0.88	0.99	1.00	0.88	0.99	0.87	0.99

The XGBoost models trained on feature set "X", "n2v-1" and "n2v-5" achieve 1.00 top-recall scores. They successfully ranked all suspicious accounts at the top 20% accounts. The lowest performance comes from "X_std": 0.42 top-recall; however, this score is still much higher than the GCN-Vanilla model that is trained "X_std".

Since the top-recall scores are calculated on the 20% accounts, this might be a large tolerance for the three models that achieve almost perfect results. Table 6.8 summarises their performance on the top 1007 accounts (the number of all suspicious account in the dataset).

Table 6.8: Performance of the best XGBoost models on their top 1007 accounts.

	X	n2v-1	n2v-5
Count	968	990	**1007**
Top-recall	0.96	0.98	**1.00**

It shows that "n2v-5" indeed ranks all suspicious accounts at the top, successfully separating the two classes. Also, given that all six models trained on n2v feature sets achieve top-recall at least 0.77 ("n2v-2"), the success of the combination of node2vec and XGBoost shows the XGBoost has the capability of exploiting graph structures.

We concluded from Chapter 5 (Classical Methods) that XGBoost is a very powerful classifier, and from Figure 5.5 (Feature Importance) that XGBoost is good at exploiting graph structures. A node feature set that comprehensively represents graph structures would fit XGBoost's strength; the output of node2vec can be such a node feature set. The combination of XGBoost and node2vec is a combination of a powerful classifier that has strength in exploiting graph structures and a graph embedding method that is capable of representing graph structures in node features. The complementary natures of the two methods explain the success of their combination.

Chapter 7

Discussion

This chapter presents the discussion about the two contributions this thesis made. One is the simulated dataset (Section 7.1.1) and the other is all classifiers investigated (Section 7.1.2). And in Section 7.1.3, we discuss the connections between the data and the classifiers. Section 7.2 shows the conclusion of the thesis. Section 7.3 provides possible directions for future work.

7.1 Discussion

7.1.1 Data

Randomness Many parts of the data simulation process involve randomness. First, the initial deposits part. The *accounts* configuration sets a range within which the simulator assigns a random number to each account as its deposit. Second, the amount of money in each transaction is randomly selected for both normal transactions and suspicious transactions. However, this is usually not the case. It is a common practice that financial institutes closely monitor transactions whose amount exceeds a threshold, say €10K. And many money-laundering transactions are made deliberately a little bit lower than the threshold, say €9.5K Euro, to avoid being flagged as suspicious. This operation, amongst other "thoughtful" actions, is ignored; the simulator simplifies them with randomness.

Suspicious Patterns The pre-defined suspicious patterns used by the simulator are common even in normal financial activities. For example, *fan_out*, where a single sender sends money to multiple beneficiaries, could apply to many accounts as people make payment to

many facilities and/or transfer money to different individuals. In the simulated datasets, these simple patterns could be generated in the process of generating the connection graph. These innocent transactions and accounts that accidentally follow the suspicious patterns would have the same sub-graph structures as real suspicious ones; the only difference between them is that real ones are labelled as suspicious while innocent ones are not.

7.1.2 AML Classifiers

This thesis studies the Anti-Money Laundering (AML) problem. AML is considered a binary classification problem: to accurately predict the types of bank accounts, either suspicious or normal. Specifically, this thesis verifies the effectiveness of classical classifiers and explores the applicability of Graph Convolutional Networks (GCNs) on the AML problem.

Classical Methods Five classical classifiers are verified, two of them are linear models: Support Vector Machine and Logistic Regression; three of them are tree-based models: Decision Tree, Random Forest and XGBoost. The two linear models are not able to separate the two classes, only performing slightly better than a trivial random guesser. By contrast, the three tree-based classifiers achieve satisfying performances in terms of ROC-AUC and top-recall scores. Among the five classifiers, XGBoost appears to be the best classifier on the AML dataset. The feature importance scores produced by XGBoost show that features related to spatial and temporal graph structures are more important than those related to transaction statistics.

When using the account features that are produced by a graph embedding method, namely node2vec, XGBoost even achieves the perfect result: ranking all 1007 suspicious accounts together at the top, achieving top-recall and ROC-AUC of 1.00. This result further illustrates the capabilities of XGBoost in exploiting and learning from graph structures.

GCN Methods To apply GCNs to the AML problem, a graph where nodes represent accounts and edges represent transactions is constructed. This thesis uses GCNs as binary classifiers, which output the probabilities of being suspicious and of being normal for every node. Three GCN variants, namely GCN-Vanilla, GCN-EF and GAT, are applied to the AML dataset to investigate their applicability.

Individual GCN-Vanilla models do not perform well on the AML dataset, but they reserve the potential of forming a powerful ensemble. GCN-EF introduces the money amounts of transactions as edge weights into GCN-Vanilla but does not improve the performance. Instead, GCN-EF models are more unstable comparing to GCN-Vanilla models and produce inferior performance. GAT, with a more complicated graph structure, achieves the best performance among the three variants. It shows the capability to separate the two classes to a certain extent.

For GCNs, depths and kernel sizes are essential to their performance. The depth of a GCN determines how far a node is able to receive messages from. Shallow networks focus on local graph structures while deep networks see more parts of the graph. In the AML problem, the information in the near neighbourhood is more important than that of the farther and larger neighbourhood. The discovery that shallow networks perform better supports the hypothesis. The kernel size of a GCN implies the complexity of a model. Increasing the complexity of GCN-Vanilla models increase their performance. An even more complicated model GAT with multiple heads achieves better results. These two facts show that the GCN application on the AML problem is sensitive to network complexity and is able to gain from high complexity.

From the performance of both classical classifiers and GCNs, the following discoveries can be seen. (1) The standardised property node feature ("X_std") is only useful for linear models. When "X_std" is applied on XGBoost and GCN, the results are significantly inferior that its non-standardised version "X". (2)Graph structures outweigh transaction statistics (e.g. the amount in each transaction) in the AML problem. Both the best XGBoost model and GCN model are trained on node2vec features that purely contain the structural graph information.

7.1.3 Connections Between Data and Classifiers

Both the processes of generating AML datasets and applying classifiers to the datasets are done in the thesis, yet they are done independently. The goal of the dataset generating step is to generate AML transactions as realistic as possible by, for example, incorporating public collective transaction data statistics and integrating known money laundering patterns. For the classifiers, they use the features extracted from the AML dataset as if the dataset generating process is unknown. The first type of features extracted ("X" and "X_std") from the transaction graph is the properties of the graph; the properties are extracted regardless of the design of the transaction data. The second type of features ("n2v" series) is the output node embeddings of node2vec. Although node2vec takes advantages of graph structures, no extra information about the suspicious patterns in the graph is leaked to node2vec. This is the independent part of the relationship between the data generation and classifier applications.

Their relationship also has a dependent part. Classifiers could achieve different performance on different AML datasets. If classifiers are applied to various datasets with different characteristics of the dataset highlighted, the results could show which aspects of the dataset are essential to the AML problem. And generating diversified datasets is what a data generator capable of. Yet, this work has not been done in this thesis. Possible highlights could be certain types of suspicious patterns or the volume of transactions in such patterns. More complex highlights could also be different degree distributions in

normal transactions; with this highlight, the effect of the thickness of the disguise provided by normal transactions could be investigated.

7.2 Conclusion

This thesis investigated the effectiveness of classical methods and state-of-art GCN methods on the AML problem. Due to the unavailability of public transaction data, this thesis improved an existing data simulator to generate realistic AML transaction data for further forensic analysis. For this AML problem, the classical method XGBoost performed very well, while GCN methods were not good as expected. And GCNs were more complex than XGBoost. GCNs involve many components and hyper-parameters, making them hard to train; XGboost, by contrast, is easier to produce high-quality models and needs less time to train and run. On this AML problem represented by the synthetic transaction data, the state-of-art GCN methods were not as powerful as classical methods.

7.3 Future Work

For data availability, the proposed data simulator AMLSim-R can be further improved. Adding suspicious patterns into the data simulator manually is an endless work. Designing a simulator that is able to imitate the financial activities in a real financial system itself would be a new direction of addressing the data availability problem. Datasets generated by such simulators would be more realistic and not violate any legislation.

For novel method applications, comparing to XGBoost, GCNs have not yet achieved competitive results. The following directions can be meaningful future work.

1. The amount of money involved in a transaction is a piece of extra information. The current work normalises the money amount in a relatively easy way, which turns out not to be effective. A new strategy of using this information, for example, diluting the overwhelming normal/suspicious transaction ratio, so that suspicious transactions have a role to play.

2. Building GCN ensembles. The current work has shown that different GCN models that are trained on different node features have the tendency to correctly predict different sets of accounts. Therefore, finding the node feature sets that cover different aspects of the AML problem and designing a voting strategy accordingly can be a promising direction.

Another promising direction to address the AML problem is using unsupervised learning methods. As in real-life transaction data with a large scale, the number of known labels

for money laundering activities can be much lower than 1%, which results in insufficient supervision for model learning. Therefore, unsupervised learning such as clustering and anomaly detection could be more meaningful for real AML applications.

Appendices

Appendix A

Supplementary Material

Table A.1: Hyper-parameter search space for Support Vector Machine. "C" is the regularisation parameter; "max_iter" is the maximum number of training iterations; "tol" is the tolerance for stopping criteria.

Hyper-parameter	methods	low	high
C	uniform	0.0	5.0
max_iter	uniform	1000	5000
tol	log uniform	1×10^{-4}	1×10^{-2}

Table A.2: Hyper-parameter search space for Logistic Regression. "warm_start" defines whether to initialise the model with previous solution; "fit_intercept" defines whether to include a bias to the model; "C" is the regularisation parameter; "max_iter" is the maximum number of training iterations; "tol" is the tolerance for stopping criteria.

Hyper-parameter	methods	low	high	options
warm_start	choice	-	-	[True, False]
fit_intercept	choice	-	-	[True, False]
C	uniform	0.0	5.0	-
max_iter	uniform	100	1000	-
tol	log uniform	1×10^{-5}	1×10^{-1}	

Table A.3: Hyper-parameter search space for Decision Tree. "criterion" is the criterion to measure the quality of a split; "splitter" is the method used to split a node; "max_depth" is the maximum depth of a tree; "max_features" is the maximum number of features to be considered when searching for the best split; "min_samples_leaf" is the minimum the number of samples needed to be in a leaf node.

Hyper-parameter	methods	low	high	options
criterion	choice	-	-	["gini", "entropy"]
splitter	choice	-	-	["best", "random"]
max_depth	uniform	6	20	-
max_features	uniform	0.1	0.9	-
min_samples_leaf	uniform	1	30	-

Table A.4: Hyper-parameter search space for Random Forest. "n_estimators" is the number of trees in the model; "max_depth" is the maximum depth of a tree; "max_features" is the maximum number of features to be considered when searching for the best split; "min_samples_leaf" is the minimum the number of samples needed to be in a leaf node.

Hyper-parameter	methods	low	high
n_estimators	uniform	10	300
max_depth	uniform	6	20
max_features	uniform	0.1	0.9
min_samples_leaf	uniform	1	30

Table A.5: Hyper-parameter search space for XGBoost. "boost" defines the boost to be used in the model; "n_estimators" is the number of trees in the model; "gamma" is the minimum loss reduction needed to make a further node split; "max_depth" is the maximum depth of a tree; "learning_rate" is the boosting learning rate.

Hyper-parameter	methods	low	high	options
booster	choice	-	-	["bgtree", "gblinear", "dart"]
n_estimators	uniform	0.1	0.9	-
gamma	uniform	10	200	-
max_depth	uniform	6	20	-
learning_rate	log uniform	1×10^{-4}	1×10^{-1}	-

Table A.6: Hyper-parameter settings of different node2vec (n2v) versions of node features. "dim" is the dimension of output features; "n_iter" is the number of iterations that node2vec run; "l_walk" is the length of each walk; "n_walk" is the number of walk.

Feature Set Name	dim	n_iter	l_walk	n_walk
n2v-1	50	50	80	10
n2v-2	30	30	50	10
n2v-3	30	30	80	10
n2v-4	30	30	100	10
n2v-5	30	30	50	30
n2v-6	30	30	50	50

Table A.7: Hyper-parameter search space for GCN-vanilla and GCN-EF. "kernel size" is the number hidden units in each GCN layer; "dropout" is the probability with which an element of its input to be zeroed; "depth" is the number of GCN layers in the model; "learning_rate" is the learning rate for SGD; "weight_decay" is the penalty factor applying to the model weights.

Hyper-parameter	methods	low	high	options
kernel size	choice	-	-	[16,32,64]
dropout	uniform	0.1	0.9	-
depth	uniform	1	20	-
learning_rate	log uniform	1×10^{-4}	1×10^{-1}	-
weight decay	log uniform	5×10^{-5}	5×10^{-4}	-

Table A.8: The base hyper-parameter setting."lr" represents the learning rate; "depth" denotes the number of convolutional layers in a GCN network excluding the input layer; "hidden unit" is the number of hidden units in a GCN layer, except for the output layer where is the number is statically set to 2, representing the two classes; "dropout" defines the probability with which an element of its input to be zeroed.

HP	lr	depth	hidden unit	dropout
Values	0.01	2	32	0.5

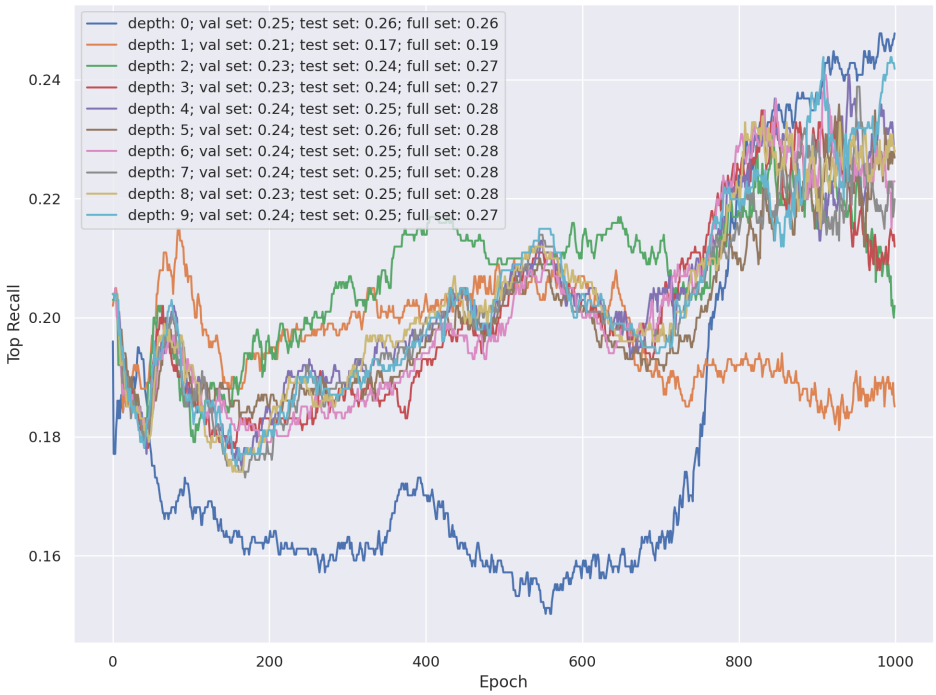


Figure A.1: Performance of GCN-EF with different depths on the validation set.

Figure A.2: Performance of GCN-EF with different kernel sizes on the validation set.

Bibliography

Abdallah, A., Maarof, M. A., and Zainal, A. Fraud detection system: A survey. *Journal of Network and Computer Applications*, 68:90–113, 2016.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Bergstra, J., Yamins, D., and Cox, D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123. PMLR, 2013.

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. *Classification and regression trees*. CRC press, 1984.

Bresson, X. and Laurent, T. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

Burt, R. S. *Structural holes: The social structure of competition*. Harvard university press, 2009.

Butaru, F., Chen, Q., Clark, B., Das, S., Lo, A. W., and Siddique, A. Risk and risk management in the credit card industry. *Journal of Banking & Finance*, 72:218–239, 2016.

Cawley, G. C. and Talbot, N. L. On over-fitting in model selection and subsequent selection bias in performance evaluation. *The Journal of Machine Learning Research*, 11:2079–2107, 2010.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

Chen, J. AMLSim-R (Realistic), 2021. URL https://github.com/CCC-S/AMLSim-R.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

Chen, Z., Teoh, E. N., Nazir, A., Karuppiah, E. K., Lam, K. S., et al. Machine learning techniques for anti-money laundering (AML) solutions in suspicious transaction detection: a review. *Knowledge and Information Systems*, 57(2):245–285, 2018.

Colladon, A. F. and Remondi, E. Using social network analysis to prevent money laundering. *Expert Systems with Applications*, 67:49–58, 2017.

Cortes, C. and Vapnik, V. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

De Nooy, W., Mrvar, A., and Batagelj, V. *Exploratory social network analysis with Pajek: Revised and expanded edition for updated software*, volume 46. Cambridge University Press, 2018.

Drężewski, R., Sepielak, J., and Filipkowski, W. The application of social network analysis algorithms in a system supporting money laundering detection. *Information Sciences*, 295:18–32, 2015.

Dutch Government. FIU-The Netherlands Annual Review 2017, 2017. URL https://www.fiu-nederland.nl/sites/www.fiu-nederland.nl/files/documenten/7238-fiu_jaaroverzicht_2017_eng_web_1.pdf. (Accessed: 2020-11-13).

Dutch Government. FIU-The Netherlands Annual Review 2018, 2018. URL https://www.fiu-nederland.nl/sites/www.fiu-nederland.nl/files/documenten/fiu-the_netherlands_annual_report_2018.pdf. (Accessed: 2020-11-13).

Dutch Government. FIU-The Netherlands Annual Review 2019, 2019. URL https://www.fiu-nederland.nl/sites/www.fiu-nederland.nl/files/documenten/fiu-nederland_jaaroverzicht_2019_en_0.pdf. (Accessed: 2020-11-13).

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

European Commission. Directive (EU) 2015/849 of the European Parliament and of the Council of 20 May 2015 on the prevention of the use of the financial system for the purposes of money laundering or terrorist financing, amending Regulation (EU) No 648/2012 of the European Parliament and of the Council, and repealing Directive 2005/60/EC of the European Parliament and of the Council and Commission Directive 2006/70/EC (Text with EEA relevance). *Official Journal of the European Union*, pages 73–117, 2015.

European Commission. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal L110*, 59:1–88, 2016.

Europol. From Suspicion To Action - Converting Financial Intelligence Into Greater Operational Impact, 2017. URL https://www.europol.europa.eu/sites/default/files/documents/ql-01-17-932-en-c_pf_final.pdf. (Accessed 2021-03-23).

Eurostat. *Money laundering in Europe - 2013 edition*. The European Commission, 2013.

FATF. Money Laundering Risks Arising from Trafficking in Human Beings and Smuggling of Migrants, 2011.

Fawcett, T. An introduction to ROC analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

Friedman, J. H. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

Goyal, P. and Ferrara, E. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

Ho, T. K. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

Hu, Y., Seneviratne, S., Thilakarathna, K., Fukuda, K., and Seneviratne, A. Characterizing and detecting money laundering activities on the bitcoin network. *arXiv preprint arXiv:1912.12060*, 2019.

Jaccard, P. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2): 37–50, 1912.

Jain, R., Kasturi, R., and Schunck, B. G. *Machine vision*, volume 5. McGraw-hill New York, 1995.

Jullum, M., Løland, A., Huseby, R. B., Ånonsen, G., and Lorentzen, J. Detecting money laundering transactions with machine learning. *Journal of Money Laundering Control*, 2020.

Khandani, A. E., Kim, A. J., and Lo, A. W. Consumer credit-risk models via machine-learning algorithms. *Journal of Banking & Finance*, 34(11):2767–2787, 2010.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Kleinbaum, D. G., Dietz, K., Gail, M., Klein, M., and Klein, M. *Logistic regression*. Springer, 2002.

Konečnỳ, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., and Bacon, D. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

Lloyd, S. Least squares quantization in PCM. *IEEE transactions on information theory*, 28(2):129–137, 1982.

Lopez-Rojas, E., Elmir, A., and Axelsson, S. PaySim: A financial mobile money simulator for fraud detection. In *28th European Modeling and Simulation Symposium, EMSS, Larnaca*, pages 249–255. Dime University of Genoa, 2016.

Lopez-Rojas, E. A. and Axelsson, S. Money laundering detection using synthetic data. In *Annual workshop of the Swedish Artificial Intelligence Society (SAIS)*. Linköping University Electronic Press, Linköpings universitet, 2012.

Maaten, L. v. d. and Hinton, G. Visualizing data using t-SNE. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

Molloy, I., Chari, S., Finkler, U., Wiggerman, M., Jonker, C., Habeck, T., Park, Y., Jordens, F., and van Schaik, R. Graph analytics for real-time scoring of cross-channel transactional fraud. In *International Conference on Financial Cryptography and Data Security*, pages 22–40. Springer, 2016.

Nelder, J. A. and Wedderburn, R. W. Generalized linear models. *Journal of the Royal Statistical Society: Series A (General)*, 135(3):370–384, 1972.

Page, L., Brin, S., Motwani, R., and Winograd, T. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau,

D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Perols, J. Financial statement fraud detection: An analysis of statistical and machine learning algorithms. *Auditing: A Journal of Practice & Theory*, 30(2):19–50, 2011.

Provost, F. and Kohavi, R. Glossary of terms. *Journal of Machine Learning*, 30(2-3): 271–274, 1998.

Rawat, W. and Wang, Z. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

Reuters. Dutch bank ING fined $900 million for failing to spot money laundering, 2018. URL https://www.reuters.com/article/us-ing-groep-settlement-money-laundering-idUSKCN1LK0PE. (Accessed: 2021-03-23).

Rohit, K. D. and Patel, D. B. Review on detection of suspicious transaction in anti-money laundering using data mining framework. *International Journal for Innovative Research in Science & Technology*, 1(8):129–133, 2015.

Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Salehi, A., Ghazanfari, M., and Fathian, M. Data mining techniques for anti money laundering. *International Journal of Applied Engineering Research*, 12(20):10084–10094, 2017.

Sangers, A., van Heesch, M., Attema, T., Veugen, T., Wiggerman, M., Veldsink, J., Bloemen, O., and Worm, D. Secure multiparty PageRank algorithm for collaborative fraud detection. In *International Conference on Financial Cryptography and Data Security*, pages 605–623. Springer, 2019.

Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Svetnik, V., Liaw, A., Tong, C., Culberson, J. C., Sheridan, R. P., and Feuston, B. P. Random forest: a classification and regression tool for compound classification and QSAR modeling. *Journal of chemical information and computer sciences*, 43(6):1947–1958, 2003.

The Washington Post. Goldman Sachs fined record $2.9 billion to resolve 1MDB

bribery scheme, 2020. URL https://www.washingtonpost.com/business/2020/10/22/goldman-sachs-1mdb-record-fine. (Accessed: 2021-03-23).

United Nations. United Nations Convention Against Illicit Traffic in Narcotic Drugs and Psychotropic Substances, 1998. URL https://www.unodc.org/pdf/convention_1988_en.pdf. (Accessed: 2021-03-23).

United Nations. Money-Laundering Overview, 2021. URL https://www.unodc.org/unodc/en/money-laundering/overview.html. (Accessed: 2021-04-02).

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., Xiao, T., He, T., Karypis, G., Li, J., and Zhang, Z. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. *arXiv preprint arXiv:1909.01315*, 2019.

Wasserman, S., Faust, K., et al. Social network analysis: Methods and applications. 1994.

Weber, M., Chen, J., Suzumura, T., Pareja, A., Ma, T., Kanezashi, H., Kaler, T., Leiserson, C. E., and Schardl, T. B. Scalable graph learning for anti-money laundering: A first look. *arXiv preprint arXiv:1812.00076*, 2018.

Weber, M., Domeniconi, G., Chen, J., Weidele, D. K. I., Bellei, C., Robinson, T., and Leiserson, C. E. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *arXiv preprint arXiv:1908.02591*, 2019.

World Bank. Gross domestic product 2019, 2021. URL https://databank.worldbank.org/data/download/GDP.pdf. (Accessed: 2021-04-02).

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

Yang, Q., Liu, Y., Chen, T., and Tong, Y. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2): 1–19, 2019.

Zhang, Y. and Trubey, P. Machine learning and sampling scheme: An empirical study of money laundering detection. *Computational Economics*, 54(3):1043–1063, 2019.

Zhang, Z., Salerno, J. J., and Yu, P. S. Applying data mining in investigating money laundering crimes. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 747–752, 2003.