Application of Boosting Algorithms for Anti Money Laundering in Cryptocurrencies

Towards Healthier Cryptocurrency Networks

Dylan Vassallo

Supervised by Dr Vincent Vella

Co-supervised by Dr Joshua Ellul

Department of Artificial Intelligence

Faculty of Information & Communication Technology

University of Malta

August, 2020

A dissertation submitted in partial fulfilment of the requirements for the degree of M.Sc. Master of Science in Artificial Intelligence .

L-Università ta' Malta

Declaration by Postgraduate Students

(a) Authenticity of Dissertation

I hereby declare that I am the legitimate author of this Dissertation and that it is my original work.
No portion of this work has been submitted in support of an application for another degree or qualification of this or any other university or institution of higher education.
I hold the University of Malta harmless against any third party claims with regard to copyright violation, breach of confidentiality, defamation and any other third party right infringement.

(b) Research Code of Practice and Ethics Review Procedures

I declare that I have abided by the University's Research Ethics Review Procedures.
As a Master's student, as per Regulation 58 of the General Regulations for University Postgraduate Awards, I accept that should my dissertation be awarded a Grade A, it will be made publicly available on the University of Malta Institutional Repository.

Faculty/Institute/Centre/School	Faculty of Information & Communication Technology
Degree	M.Sc. Master of Science in Artificial Intelligence
Title	Application of Boosting Algorithms for Anti Money Laundering in Cryptocurrencies
Candidate (Id.)	Dylan Vassallo (0043794M)

Signature of Student	______________________
Date	February 5, 2021

08.02.2018

For my Martina, your patience is divine.
No more proofreading, I promise this time.
Thank you for your love and support.

Acknowledgements

I would first like to thank my supervisors, Dr Vince Vella and Dr Joshua Ellul, whose knowledge, encouragement and support, guided me to successfully complete this research. A special thanks goes to my family and friends, who always supported me through these years of study. Last but not least, I am honoured to have been awarded a Scholarship in Blockchain and DLT, initiated by the Parliamentary Secretary for Financial Services, Digital Economy and Innovation in collaboration with the University of Malta and the Malta Information Technology Agency.

Abstract

Detecting money laundering is an essential function to protect the global economy, and it is vital to have systems and laws in place to counteract this nefarious activity. The recent emergence of cryptocurrencies has added another layer of complexity in the fight towards financial crime, while also creating an intriguing paradoxical paradigm: blockchain, the core component that underpins cryptocurrencies, functions without a central authority and offers pseudo-anonymity to its users, allowing criminals to disguise themselves amongst them, on the other hand, the openness of data, fuels the investigator's toolkit to conduct forensic examinations. Meanwhile, the application of machine learning to combat and detect these crimes by leveraging data to build more robust compliance and Anti Money Laundering (AML) systems is advancing and exhibits great potential to safeguard the economy. This study focuses on the initial stages of money laundering, primarily, the detection of illicit activities (such as scams, financing terrorism, Ponzi-schemes) on cryptocurrency infrastructures, on both an *'account'* and *'transaction'* level. The common denominator between these crypto-related crimes is money laundering. Once an unlawful user gains access to these illicitly-gained funds, the primary focus shifts into "washing" them without detection.

Utilising 4,681 Ethereum accounts and 46,564 Bitcoin transactions, we attempt to detect illicit activities using three state-of-the-art variants of gradient boosting, eXtreme Gradient Boosting (XGBoost), Light Gradient Boosting Machine (LGBM) and CatBoost. However, given the widespread issue of class imbalance in this domain and its dynamic environment that is created by the techniques employed by criminals which are continuously evolving to avoid detection, we seek to address these problems in order to mitigate the negative ramifications attributed to them. Employing Neighbourhood Cleaning Rule (NCL), Synthetic Minority Over-Sampling (SMOTE) and NCL-SMOTE as data-sampling techniques, and using our proposed innovative adaptation of XGBoost, *'Adaptive Stacked eXtreme Gradient Boosting (ASXGB)'*, developed to handle non-stationary data, we successfully reduced the impact of concept drift, an issue which is often overlooked in this domain as well as, class imbalance. LGBM obtained the highest F1-Score of 0.820 on the *'transaction'* level data, whilst XGBoost obtained the highest F1-Score of 0.983 on the *'account'* level data, with further improvements made on the *'transaction'* level data when we used data-sampling techniques. We also showed that our ASXGB was one of the fastest model to adapt to concept drift when compared against other state-of-the-art adaptive learners on the *'transaction'* level data. Based on the obtained results, the proposed approaches are highly effective in the detection of illicit activities over cryptocurrency networks, at both an *'account'* and *'transaction'* level, obtaining fewer False Positives and False Negatives rates in comparison to previous work in this domain and industry standard (up to 90% False Positives).

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ADASYN	Adaptive Synthetic Sampling	48
ADF	Augmented Dickey-Fuller test	72
ADWIN	ADaptive WINdowing	57
AML	Anti Money Laundering	xi
ANN	Artificial Neural Network	15
ARF	Adaptive Random Forest	56
ASXGB	Adaptive Stacked eXtreme Gradient Boosting	xi
AUC	Area Under the ROC curve	32
AXGB	Adaptive eXtreme Gradient Boosting	60
CNN	Condensed Nearest Neighbors	49
DT	Decision Tree	4
ENN	Edited Nearest Neighbors	50
GAN	Generative Adversarial Networks	51
GBM	Gradient Boosting Machine	78
GCN	Graph Convolutional Network	4
KNN	K-Nearest Neighbours	34
NCL	Neighbourhood Cleaning Rule	xi
NM	Near Miss	49
LGBM	Light Gradient Boosting Machine	xi
LR	Logistic Regression	32
OSS	One-Sided Selection	50
RF	Random Forest	4
RIPPER	Repeated Incremental Pruning to Produce Error Reduction	34

1 Introduction

1.1 | Anti-Money Laundering in Cryptocurrencies

The emergence of cryptocurrencies, introduced by Bitcoin back in 2009 (Nakamoto, 2009), enabled peer to peer digital money transfers across all borders, with the added advantage of doing so at a low cost. These qualities have been made possible through blockchain technology, which is the core component that underpins most cryptocurrencies. Bestowing trust among its users by virtue of being immutable, blockchain can be regarded as a digital public ledger, enabling peers within a network to execute tamper-proof transactions without the need of a third-party. This novel idea offers robustness by eliminating any single points of failure, self-sufficiency through distributed consensus, security via cryptography, and consistency due to its immutable nature. All of these characteristics packed in a practical decentralised payment network, ignited scepticism back in the early days of cryptocurrencies. Genuine concerns were being raised on the legitimacy of these virtual currencies, as they acted as a form of payment in dark-web marketplaces to buy and sell illicit goods (mail-order drugs and weapons). Criminals managed to take advantage of this technology as transactions could not be blocked, and it offered a certain level of anonymity. In addition to this exploitation, cryptocurrencies fell outside the regulatory provisions set out by governments and lawmakers (Kaplanov, 2012; Trautman, 2014).

Fast-forward to today, and thousands of virtual currencies later (Yuan and Wang, 2018), Bitcoin is ranked as the number one cryptocurrency in terms of market capitalisation ($117.81bn), where it peaked at around $238bn in the fourth quarter of 2017 [1]. Other cryptocurrencies, such as Ethereum expands on the functionality provided by Bitcoin

[1] BTC Market Cap: `https://www.statista.com/statistics/377382/bitcoin-market-capitalization`

due to the introduction of *smart contracts*, permitting developers to write code using a Turing-complete programming language to create decentralised applications on the blockchain (Buterin et al., 2014). With this technology continually evolving and improving, the tone towards cryptocurrencies has shifted. The number of businesses building products and services based on blockchain and cryptocurrencies is ever increasing, especially in financial services such as remittance and online payments (Frizzo-Barker et al., 2020; Yuan and Wang, 2018).

Despite adoption, there is still a certain level of scepticism especially towards Bitcoin, in fact, multiple high profile people described it to be; a speculative investment bubble, being a scam and dominated by illegal endeavours [2] (Frizzo-Barker et al., 2020). The statement of cryptocurrencies being dominated by illegal endeavours is not entirely unfounded, although criminal activity does not dominate the crypto space, it still accounts for about 1-2% of the overall activity [3] (Chainalysis, 2020; Elliptic, 2020; Weber et al., 2019). These illicit activities include, but are not limited to; Ponzi-schemes, scams, stolen funds, ransomware and financing terrorism. In all of these crypto crimes, money laundering is the common denominator, as once a criminal gets hold of these illicitly-gained funds, their focus shifts to turning them into legitimate assets or cash without detection (Chainalysis, 2020). Given the rise in popularity and the exponential growth in the value of cryptocurrencies, coupled with the fear of criminals hiding behind the pseudonymous nature of this technology; governing bodies, regulators and lawmakers started to establish regulations in order to combat money laundering and other crypto crimes (European Parliament and Council, 2018; Financial Action Task Force, 2019; FinCen, 2019; Lessambo, 2020).

The 'Fifth Anti-Money Laundering Directive (5AMLD)' (European Parliament and Council, 2018) was recently adopted by the European Union to counter financing of terrorism and money laundering. The law applies to the following entities operating in one of its 28 member states: exchanges allowing its users to trade crypto-to-crypto or crypto-to-fiat (or vice-versa), and wallet service providers holding the users' private keys. These entities are obliged to apply proper customer due diligence such as KYC procedures, monitor transactions, maintain customer history and report any transactions which are deemed as suspicious, all of which needed to be inplace by the end of 2019. Similar guidelines [4] were set out by the Financial Action Task Force (FATF) (Finan-

[2] Bitcoin Bulls and Bears: `https://www.bloomberg.com/features/bitcoin-bulls-bears/`

[3] MIT-IBM AI Lab Analyzed 200,000 Bitcoin Transactions. Only 2% Were Labeled 'Illicit': `https://www.coindesk.com/mits-ai-lab-crunched-200000-bitcoin-transactions-only-2-were-illicit`

[4] New money-laundering rules change everything for cryptocurrency exchanges: `https://www.technologyreview.com/2019/08/15/102778/new-money-laundering-rules-change-everything-for-cryptocurrency-exchanges/`

cial Action Task Force, 2019) to its member jurisdictions (including; the United Kingdom and the United States) to regulate cryptocurrency marketplaces. In addition, the United States are also subject to other guidelines (FinCen, 2019) set out by the Financial Crimes Enforcement Network (FinCEN) to regulate business models involving cryptocurrencies. Failure to comply with these regulations can result in hefty fines, revoking of licenses, or even worse shutting down of businesses. Abiding by these stringent rules is crucial for businesses since having a lack of Anti Money Laundering (AML) controls can come at a hefty price. These negative ramifications impose liable entities to implement systems which can monitor and detect suspicious behaviour in cryptocurrencies. The upside is that the environment is inherently transparent, as each transaction is persisted in a public ledger. This allows investigators to analyse money laundering ecosystems at a high level, so as to extract insights from these detected illicit networks/rings. All of these can be demanding or near impossible to employ when monitoring traditional fiat finance, as multiple third parties may be involved (Alsuwailem and Saudagar, 2020; Samanta et al., 2019; Singh and Best, 2019).

In order to do so, the right tools must be in place as transactions are broadcasted to the blockchain at a high volume and velocity, making this task infeasible to be handled manually. The application of machine learning can circumvent these hurdles, however, the detection of nefarious activities on blockchain networks is non-trivial since illicit transactions or accounts are very few when compared to licit ones (Chainalysis, 2020; Elliptic, 2020; Weber et al., 2019), and this could hinder the overall performance (Abd Elrahman and Abraham, 2013; Bartoletti et al., 2018). Another additional layer of complexity is compounded by the fact that the underlying techniques employed by criminals are continually evolving, hence making the environment dynamic and continuously changing (van der Veer, 2019; Weber et al., 2019).

1.2 | Motivation

As of May 2020, the total number of transactions processed for the top two cryptocurrencies in terms of market capitalisation, Bitcoin [5] and Ethereum [6] was - approximately 535M, and approximately 721M transactions, respectively. Given the high volume and velocity, resources are required to be able to monitor these transactions so as to detect suspicious activity on the network. It is extremely improbable that this task can be achieved only through manual work; hence, the need for automated detection sys-

[5] Bitcoin Charts: `https://www.blockchain.com/charts`
[6] Etherscan: `https://etherscan.io/txs`

tems is required, as a tool to aid stakeholders. Autonomously detecting anomalous behaviour or transactions, to combat money laundering and other white-collar financial crimes, has been often employed in various financial industries dealing with transactions (Alsuwailem and Saudagar, 2020). These systems could also be implemented to work with cryptocurrencies, by utilising the publicly available data on the blockchain and incorporating it to work with machine learning, thus reducing the manual work required.

The application of deep learning has been investigated in the detection of cryptocurrency crimes (Nilsen, 2019; Weber et al., 2019). Weber et al. (2019) investigated whether a Graph Convolutional Network (GCN) can be effective in identifying transactions linked to nefarious activities. Many researchers (Baek et al., 2019; Bartoletti et al., 2018; Lee et al., 2020; Lin et al., 2019; Toyoda et al., 2017; Weber et al., 2019; Zola et al., 2019) also investigated the use of Random Forest (RF), which is an ensemble of decision trees and an extension of bootstrap aggregation. RF has shown to be appealing in this domain due to its effectiveness, ease of use and explainability. In a recent study (Weber et al., 2019), it was shown that RF can outperform deep learning approaches when identifying suspicious transactions executed on the Bitcoin network. Decision Tree (DT) based gradient boosting, also an ensemble of decision trees, have been investigated (Chen et al., 2018; Farrugia et al., 2020; Harlev et al., 2018; Lin et al., 2019; Sun Yin et al., 2019; Toyoda et al., 2017; Zola et al., 2019) to detect various crypto-related crimes. Farrugia et al. (2020) highlighted that gradient boosting can effectively detect suspicious behaviour on the Ethereum blockchain, when they set out to investigate the detection of illicit activities on an account-level.

Current literature highlights two challenges that make the problem of detecting illicit activity on the blockchain hard to solve, having a skewed class distribution and concept drift (Bartoletti et al., 2018; Lee et al., 2020; Lin et al., 2019; Weber et al., 2019; Zola et al., 2019). There is also the question of *"Which model is the most suitable to detect licit-or-illicit activities on blockchain networks adequately?"*. From current literature, it is evident that tree-based ensembles are one of the most widely used models in this domain. Even though gradient boosting algorithms were examined head to head against RF in detecting crypto crime (Lin et al., 2019; Sun Yin et al., 2019; Toyoda et al., 2017), it is still unclear which tree-based ensemble is more effective in the context of this problem. Due to the lack of rich datasets, hyperparameter optimisation, statistical hypothesis testing and the lack of a systematic evaluation of various state-of-the-art gradient boosting algorithms to detect illicit activities on the blockchain, we decided to confront this gap in research. Furthermore, class imbalance needed to be addressed as this characteristic is widespread in this type of problem. In previous studies, various data-sampling techniques were implemented to counteract this (Bartoletti et al., 2018; Sun Yin et al., 2019; Toyoda et al., 2017), however, the techniques employed were either non-heuristic approaches or fell short in comparing multiple heuristic approaches. Another shortcoming is that the effect of non-stationarity and shifting concepts is often overlooked in this domain, and we intend to shed light on some techniques which may help to mitigate this problem. Literature shows that these two challenges still pose an active area of research.

Our primary interest was to investigate these shortcomings in the context of detecting illicit behaviour on the blockchain at an account and transactional level using decision-tree based gradient boosting as models. Also, these proposed supervised gradient boosting algorithms were trained and evaluated by utilising a rich dataset(s). For instance, the use of a graph-structured transactional dataset was utilised when employing our models at a transactional-level. Graph-structures can be significant when combating money laundering (Gaihre et al., 2019; Samanta et al., 2019; Weber et al., 2019; Zola et al., 2019), as this criminal activity involves multiple parties; hence the relationships (payment flows) between data points needs to be examined. If we are able to detect suspicious activity at a transactional level effectively, one can later examine how these illicitly-gained-funds move from one account to another through the use of this graph structure. It would then be a matter of pinpointing criminal networks/rings, and identifying how these illicit funds are "washed" in order to appear legitimate.

The following motivations fuel our interest to detect illicit activities on the blockchain by employing machine learning.

- Given that recent regulations are being appended in various jurisdictions, we believe that the right tools should be available to aid stakeholders in carrying out the tasks needed to abide by these policies.

- Preventing these illicit activities from being carried out, not only safeguards its users, who are the primary victims of these criminal acts but also, promotes a certain level of assurance between service providers and their customers. This assurance is, therefore, mutually beneficial as it encourages more usage from the users' end and ultimately improves business profitability.

- We believe that having healthier blockchain networks will improve the reputability for this novel technology which in turn, will encourage its adoption thus, having the potential to positively alter the financial system as we know it.

We advocate that the utilisation of boosting algorithms to create more efficient detection systems may aid (i) Law enforcement agencies by reducing the manual work, and provide insights as a basis for follow up investigations (ii) Services/Businesses operating within jurisdictions that are obliged to follow money laundering regulations, by providing an automated detection tool helping these stakeholders to report suspicious behaviour (iii) Researchers and journalist which may be interested in examining the flows within the network as a means of discovering suspicious patterns such as shady criminal rings, and warn the public (iv) AML Compliance Departments, by providing the automated tools necessary to reduce the manual labour required which in turn, decreases the cost of resources and time. Resources could be shifted elsewhere such as, follow-up investigations from the suspected suspicious behaviour detected.

1.3 | Aims & Objectives

This study aims to investigate and improve on existing approaches to detect illicit behaviour on the blockchain, at an account and transaction level. We define our aim by the following research question:

Can we detect licit and illicit activities on blockchain networks at an account and transaction level through data-sampling techniques and handling concept drift?

The following objectives have been outlined, as a means of addressing this aim:

1. *Compare the selected state-of-the-art algorithms against each other, as well as against a chosen benchmark, to determine which model is most effective in the context of identifying licit-or-illicit activities on blockchain networks, at an account and transaction level.*

2. *Improve the detection of licit-or-illicit activities for the selected state-of-the-art algorithms, through the adaptation of data sampling techniques, while also identifying which approach works best.*

3. *Improve the detection of licit-or-illicit activities at a transactional-level, on state-of-the-art algorithms by handling concept drift more effectively in order to minimise performance degradation over time, thus enabling real-time transaction monitoring of cryptocurrency.*

1.4 | Contributions

Since there are a large number of distinct cryptocurrencies (Yuan and Wang, 2018), we primarily focused on evaluating our solution to detect illicit Bitcoin transactions and illicit Ethereum Accounts. Given the time constraints, we were required to limit the investigation to Bitcoin and Ethereum, which were chosen since they are the two most used cryptocurrencies. Utilising two cryptocurrency datasets, the *Elliptic dataset* (Elliptic, 2020; Weber et al., 2019) and *Ethereum Illicit Accounts dataset* (Farrugia et al., 2020), we were able to showcase the results and potential for the proposed approach.

In our proposed solution, we improve on state-of-the-art models in the context of identifying illicit cryptocurrency accounts and transactions, by effectively addressing the primary challenges, namely, class imbalance and concept drift. In our work we replicated and extended recent published literature in this domain, resulting in the proposal of an innovative adaptation of XGBoost, that we named as Adaptive Stacked eXtreme Gradient Boosting (ASXGB).

With respect to the state-of-the-art results, when compared on the *'Elliptic'* (Weber et al., 2019) and *'Ethereum Illicit Accounts'* (Farrugia et al., 2020) datasets, we showed an improvement in terms of F1-Score. More specifically, we obtained an improvement of +2.4% in F1-Score when comparing the proposed LGBM model against the top performing model in Weber et al. (2019) study (Random Forest). We also improved on the best performing model in Farrugia et al. (2020) study (XGBoost using GridSearch) by +2.3% in F1-Score when compared against the proposed XGBoost model tuned using Tree-structured Parzen Estimator (TPE). We showed that by applying data-sampling techniques (SMOTE, NCL, NCL-SMOTE), it further improved the overall performance in terms of F1-Score, recall and precision. We also show that our proposed ASXGB outperformed another state-of-the-art adaptation of XGBoost (Montiel et al., 2020) in terms of F1-Score and ranked slightly lower when compared to an adaptation of Random Forest used to handle non-stationary data (Gomes et al., 2017). Although F1-Score was slightly lower in ASXGB, the recall (+9.3% from the results obtained by Weber et al. (2019)) is still improved, meaning that when concept drift occurs, it does not suffer from false negatives and is the fastest to adapt to change.

1.5 | Document Structure

The remainder of this document covers the following; Chapter 2 serves as a basis for the background information required to understand the work presented in this study.

Chapter 3 provides a detailed review of previous research covering (but not limited to) money laundering detection in financial systems, handling class imbalance, hyperparameter optimisation and handling drifting concepts. Chapter 4 defines the methodology and the design for the proposed solution. Chapter 5 outlines the evaluation framework employed and results, followed by a discussion on the corresponding outcomes in Chapter 6. Lastly, the final chapter includes potential future work stemming from this research and concluding remarks.

2 Background

2.1 | Money Laundering

Since money laundering affects the global economy and its security (Schneider and Windischbauer, 2008; Schott, 2006), it is crucial to have tools and laws in place to prevent this illegal activity from happening (Lessambo, 2020). The process of laundering money is split into three phases, coined as; *Placement*, *Layering* and *Integration*. The placement phase involves money obtained from illicit activities, which is injected into the financial system by imitating ordinary business transactions. Placement techniques include but, are not limited to, moving money through shell companies or cash-intensive businesses. In the layering phase, these illicit funds go through multiple transactional hops as a means of concealing their original source, and this is usually carried out through multiple on-shore and off-shore bank accounts. In the final phase, these funds are transferred back to the source in the form of assets, whereby money is moved by imitating ordinary business transactions, similar to the placement phase (Lessambo, 2020; Schneider and Windischbauer, 2008).

Money laundering and other financial crime are a global threat to the world economy and so, laws and guidelines are reenacted to limit the damage caused by these nefarious activities. Failure to comply can result in infringement penalties, for instance, back in 2012, HSBC was held accountable by US authorities for failing to apply Anti Money Laundering (AML) measures, paying \$1.9bn due to allegations that their lax controls enabled laundering of drug money (Yicheng, 2015). Similarly, Danske bank was found to have enabled an estimated \$230bn in suspicious transactions coming from various countries such as Russia, British Virgin Island and Britain. Failure to screen clients caused the bank its reputation, the consequence of several billion-dollar fines, and criminal charges against their employees. Moreover, this event, which is considered to be one of the biggest laundering scandals in Europe, uncovered potential loopholes in the banking legislation set up by the European Union (Yeoh, 2019).

Since the creation of cryptocurrencies (Nakamoto, 2009) back in 2008/2009, criminals found ways to exploit the Blockchain, to conduct money laundering and other criminal activities (Adam and Maitland, 2018; Bartoletti et al., 2018; Bryans, 2014; Chainalysis, 2020; Frizzo-Barker et al., 2020; Weber et al., 2019). As this network utilises a strong cryptography to provide peer-to-peer financial transactions in a secure and verifiable manner; its pseudonymity properties made it appealing for criminals to carry out these activities (Adam and Maitland, 2018; Bryans, 2014; Mabunda, 2018). The same concepts (*Placement*, *Layering* and *Integration*) still apply when laundering illicitly gained crypto funds, however, the processes used are relatively different (Bryans, 2014; Fanusie and Robinson, 2018; Mabunda, 2018). In the placement phase, cryptocurrencies are bought (using fiat or other cryptocurrencies) from online cryptocurrency trading markets/exchanges, which lack AML measure or are unregulated. In the layering stage, crypto funds are hidden/obfuscated by engaging in fake Initial Coin Offerings or transferring funds between accounts on unregulated exchanges, in order to break the link from the original source. Another well-known method to hide these funds is the use of mixer/-tumbler services. These services which come at a cost, obfuscate the source of cryptocurrency funds, by mixing and sending coins from one address to another (also randomising transaction amounts), which will later end up in a new address controlled by the person owning the original funds. It is also common to process/mix funds on more than one service to further reduce the chances of being traced (de Balthasar and Hernandez-Castro, 2017). At the final integration phase, funds are withdrawn (using crypto ATMs, or unregulated exchanges) and presented as currency appreciation or profitable investment, which is hard to discredit given that the market is highly volatile. Alternatively, these funds could also be exchanged for goods on online marketplaces, which accept cryptocurrencies as payment. Common services/practices related to laundering crypto funds are shown in Figure 2.1.

2.2 | Blockchain and Cryptocurrency

In order to understand how money laundering and other financial crimes operate within the cryptocurrency space, one must have a basic understanding of the underlying technology behind these currencies. Blockchain technology is the core component which enables most cryptocurrencies to operate within a decentralised network. It was first envisioned by Nakamoto (2009) back in 2008/2009, and various alterations and improve-

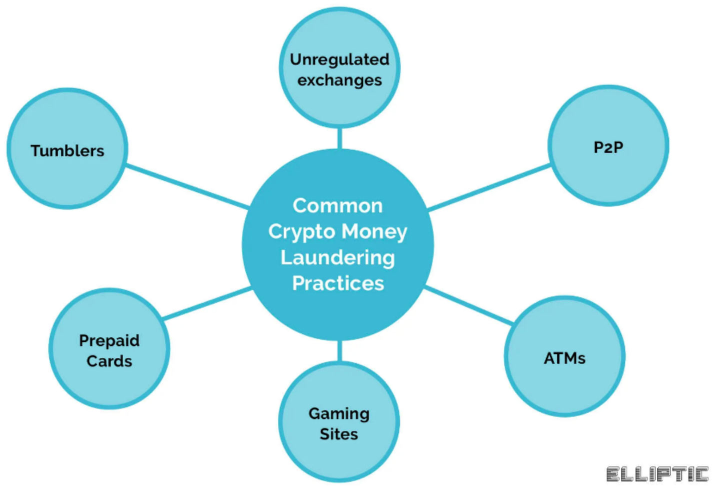


Figure 2.1: The most notable services/practices which are involved in laundering cryptocurrency funds. Source: Bitcoin Money Laundering: How Criminals Use Crypto - `https://www.elliptic.co/our-thinking/bitcoin-money-laundering`

ments were implemented along the years. These enhancements include but are not limited to; (i) additional sophistication with the introduction of smart contracts, enabling the transfers of more complex assets beyond digital payments, such as loans/bonds and equity (Blockchain 2.0) (ii) the ability to build applications outside the financial realm such as distributed governance and health systems (Blockchain 3.0) (Frizzo-Barker et al., 2020; Swan, 2015). A brief description of the various components and processes that make up the blockchain is given below. For the sake of simplicity, the components described refer to a permission-based blockchain network [1], in particular the Bitcoin network.

2.2.1 | Blockchain Technology

The main components in the blockchain infrastructure are cryptography, transactions, a secured distributed ledger and a consensus mechanism. Cryptography is used in many different ways in this infrastructure, most notably to sign (private keys) and validate (public keys) transactions broadcasted to the network. It is, in fact, cryptography that allows for pseudo-anonymity within the network, as users hide behind a generated address (can be viewed as an individual bank account). Transactions are used to send/re-

[1] There are two main types of blockchains, permissionless-based (such as Bitcoin) which publicly validates transactions, and permission-based (such as Hyperledger Fabric), where transactions are validated/executed by users approved by the owner (tend to be more centralised) (Frizzo-Barker et al., 2020)

ceive value in this distributed system, and this component holds information such as the value sent, the list of inputs and outputs, and a timestamp ('locktime'). Transactions are then grouped to form blocks, and these represent the current state of the network. These blocks are then verified by miner nodes using a concept coined as *'Proof of Work'*, where miners solve cryptographic puzzles and once solved a new block is constructed. Block miners who manage to solve the problem earn cryptocurrency, and start working on the next set of transactions (block). All confirmed blocks are persisted on a secured distributed ledger (Frizzo-Barker et al., 2020; Lee et al., 2020; Swan, 2015).

2.2.2 | Cryptocurrency

Cryptocurrencies are virtual currencies were cryptography is utilised in order to distribute and create these units (Mukhopadhyay et al., 2016), or as Nakamoto (2009) put it *"an electronic coin as a chain of digital signatures"*. The idea of having an anonymous electronic money system dates back to 1983, where Chaum (1983) developed the idea of *eCash*. Similar to cryptocurrencies, eCash employed cryptography to give a level of anonymity. However, one significant difference was that this virtual currency relied on a central authority (banks) (Mukhopadhyay et al., 2016). Bitcoin was the first-ever virtual currency successfully implemented as a decentralised monetary system (Mukhopadhyay et al., 2016; Nakamoto, 2009). Inline with the previous section, discussing the blockchain network, the functionality of these currencies works as follows; (i) a wallet is tied to a user via an address which acts as a public key (ii) a private key is then utilised to sign a transaction, providing a proof of ownership (iii) a user can send or receive cryptocurrencies via a wallet (iv) these transactions are then verified by block miners (in a *'Proof of Work'* scenario) (Mukhopadhyay et al., 2016).

Since its inception, various cryptocurrencies have been developed, where each currency has its requirements and usage. For instance, *Ethereum (ETH)* (Buterin et al., 2014), can be seen as an alternative to Bitcoin, where it allows the development of decentralised applications, *Ripple (XRP)* a pre-mined currency enables real-time settlements for banks (Schwartz et al., 2014) and *Monero (XMR)* [2] offering near-total anonymity to its users. Each currency may also adopt a specific consensus mechanism, for instance, some may use *Proof of Work* while others opt for *Proof of Stake* (Mukhopadhyay et al., 2016). While consensus mechanisms are a base layer of cryptocurrencies and therefore, are an essential function to detail, their intricacies will not be discussed further in this study, so that the focus remains on the detection of illicit activities on the blockchain.

[2] Monero Whitepaper: `https://github.com/monero-project/research-lab/blob/master/whitepaper/whitepaper.pdf`

2.2.3 | Illicit Activities on the Blockchain

The reason why criminals are attracted to conduct their crimes within blockchain infrastructures, is that there is no central authority, there are a lack of regulations (although improving in recent years) and the pseudo-anonymity that the technology offers (Adam and Maitland, 2018; Frizzo-Barker et al., 2020; Lee et al., 2020; Samanta et al., 2019). In order to be able to discern between the "good" and "bad" actors, one needs to have a clear indication of the services/organisations operating within the blockchain. These services include but are not limited to; exchanges, hosted wallets, mining pools, Ponzi-scheme, high-yield-investment-programs (HYIP), scams, terrorism-financing, ransomware, mixers, phishing schemes, fake token sales, blackmail, gambling, and dark marketplaces (Adam and Maitland, 2018; Chainalysis, 2020; Harlev et al., 2018; Weber et al., 2019). These services can be grouped into illicit or licit services, for instance, an exchange and hosted wallets can be considered as licit services, while scams and terrorism-financing as illicit ones (Weber et al., 2019); however, it is not always "black and white" as exchanges which are commonly considered as legitimate services, have been shown to be exploited by suspicious transactions (Baek et al., 2019). As stated in Section 1.1, the common denominator in cryptocurrency-related crime is money laundering, as once a criminal obtains illicitly-gained funds, the primary focus shifts to integrating these funds within the financial system. So, in a way, having systems in place to detect illicit services or transactions will help the fight against money laundering, as the flow of payments can then be analysed given the open nature of the blockchain.

2.3 | Detection via Machine Learning

In this subsection, a high-level overview of various paradigms applied in machine learning as a means of intercepting or detecting illicit financial activities is given, together with common obstacles one must handle to ensure a robust system is in place.

Money service businesses are on the rise as more and more companies are involving themselves in the processing of payments which in turn, is making the process of tracing money laundering and other financial crimes almost impossible (van der Veer, 2019). Adding cryptocurrency exchanges, new payment systems and third-party merchant accounts such as PayPal[3] and Skrill[4] into the equation, the risk of potential financial crime going undetected is more prominent than ever (van der Veer, 2019). Luckily, the use of machine learning to combat and detect these crimes, by leveraging data in or-

[3] PayPal: `https://www.paypal.com`
[4] Skrill: `https://www.skrill.com`

der to build more robust compliance and AML systems, is also gaining traction in both academia and the industry (Dorofeev et al., 2018; Jullum et al., 2020; Sun Yin et al., 2019; van der Veer, 2019; Weber et al., 2019).

One of the most primitive methods employed to detect financial crime such as money laundering, are rule-based systems, which can be viewed as a collection of IF-THEN statements (Helmy et al., 2016). In essence, these notification-based techniques operate on a set of rules with rigid thresholds, and whenever a threshold exceeds a specified limit, it is considered as suspicious. Investigators are then notified to conduct further manual inspections (Helmy et al., 2016; Jullum et al., 2020). Even though rule-based systems are still being implemented and utilised in the industry (Chandradeva et al., 2020; Helmy et al., 2016; Jullum et al., 2020), these techniques have their drawbacks, some of which include; (i) requiring expert knowledge to create new rules (ii) requiring extensive maintenance which in turn increases the cost of operation (iii) performance declines as rules are compounded (iv) prone to high false positives (Chandradeva et al., 2020; Jullum et al., 2020). Furthermore, in the case of money laundering prevention, the Financial Action Task Force is recommending to migrate from rule-based systems to a more risk-oriented approach (Financial Action Task Force, 2012; Savona and Riccardi, 2019). Due to this recommendation and other operational factors such as the need for automation, the application of machine learning is being implemented across the financial industry, to assess client risk and enhance due diligence processes (Alsuwailem and Saudagar, 2020; Savona and Riccardi, 2019). Machine learning algorithms developed to prevent these nefarious activities can be categorised into two; *Supervised Learning* and *Unsupervised Learning* (Chandradeva et al., 2020; Jullum et al., 2020).

2.3.1 | Supervised Learning

In supervised learning, the algorithm learns from a set of labelled instances, where inputs/features denoted as x are mapped to their corresponding target variable y. The objective in this type of learning, is to train the algorithm until it is adequate to approximate the mapping between the inputs to the target variables, expressed as $Y = f(X)$ (Friedman et al., 2001). As Friedman et al. (2001) stated, the objective of supervised learning is *"to predict the value of an outcome measure based on a number of input measures"*. This type of learning has been thoroughly investigated (Alsuwailem and Saudagar, 2020; Harlev et al., 2018; Sun Yin et al., 2019; Weber et al., 2019) and proved to be effective in detecting financial crime such as fraud (Chandradeva et al., 2020; Fiore et al., 2019; Lim et al., 2014) and Ponzi-schemes (Bartoletti et al., 2018; Chen et al., 2018); however, in money laundering, it is rarely the case where an organisation gets feedback whether a suspicious client was charged with an offence. Therefore, an algorithm is commonly modelled on suspicious behaviour when supervised learning is employed to detect this activity (Jullum et al., 2020).

Zhang and Trubey (2019) investigated five different supervised learning classifiers to facilitate the process of decision-making when filing Suspicious Activity Reports (SARs), by utilising data constructed from previously flagged alerts (containing both aggregated transactional and account information). These algorithms included; (i) C5.0 Decision Tree (Quinlan, 1986, 1993) - a tree-based algorithm which starts from the root node and partitions the dataset into smaller subsets (using information entropy), until a leaf node representing the final decision is reached (this process is illustrated in Figure 2.2) (ii) Random Forest (RF) (Breiman, 2001) - a collection/ensemble of decision trees (built on random subsamples) which combine the outcomes of multiple trees to reduce bias and overfitting, both of which are common when building a single deep decision tree (iii) Support Vector Machine (SVM) (Cortes and Vapnik, 1995) - an algorithm which finds an optimal boundary/hyperplane to separate data points in an N dimensional space using kernel functions (iv) Artificial Neural Network (ANN) (McCulloch and Pitts, 1990) - a model that reduces error by adjusting weights between the output(s) and input(s) using backpropagation (v) Bayes Logistic Regression. In their study, they showed that they could effectively capture rare events using supervised learning classifiers, pointing out that the performance of these models prevail given that the data is of a certain quality (accurate labels and features) (Zhang and Trubey, 2019).

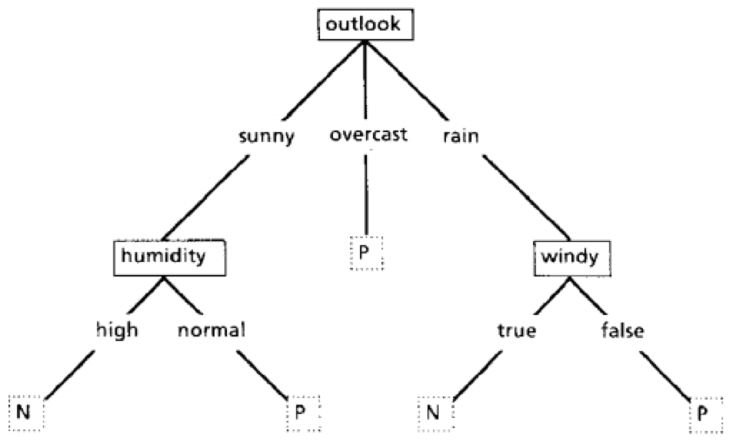


Figure 2.2: A visual representation of a simple decision tree, the first node ('outlook') is referred to as the root node. The final outcome/classification ('N' or 'P') is taken once a leaf node is reached (Quinlan, 1986).

Supervised learning techniques have also been investigated thoroughly in detecting illicit activity on blockchain networks (Baek et al., 2019; Bartoletti et al., 2018; Farrugia et al., 2020; Lin et al., 2019; Sun Yin et al., 2019; Toyoda et al., 2017; Weber et al., 2019). The downside for employing these types of learners in this particular domain (financial crime), is that most financial data is considered sensitive information; hence, it is difficult to find open labelled datasets to build detection systems (Chandradeva et al., 2020).

2.3.2 | Unsupervised Learning

Unsupervised learning is the procedure of detecting patterns without any existing labels, and the main goal is to discern patterns and associations among a set of inputs (Friedman et al., 2001). The application for this type of learning can be useful in the cryptocurrency domain, as most labelling happens by scarping information from forums and dark web marketplaces, in order to point out bad actors within a network (Bartoletti et al., 2018; Zola et al., 2019). Clustering, which is an unsupervised learning technique is another paradigm to group these instances (Harlev et al., 2018). Baek et al. (2019) employed K-means clustering so as to cluster Ethereum wallets scraped from etherscan.io (blockchain explorer). This technique splits data into *k* groups, by adjusting centroids until they are centred within the specified groups. The wallets were then grouped into seven different clusters. They noted that one of the clusters contained suspicious transactions. Another study conducted by Toyoda et al. (2018), also applied clustering techniques; however, clustering was employed in order to group multiple addresses to one entity (with a certain degree of probability). The techniques employed to cluster these samples were firstly mentioned by Androulaki et al. (2013), where they investigated K-means clustering and Hierarchical Agglomerative Clustering.

2.3.3 | Account vs Transactional Level Detection

A machine learning system built to detect criminal financial activity can be designed to identify these operations on an *account* or *transactional* level (Duman and Buyukkaya, 2008; Lim et al., 2014; Sudjianto et al., 2010). In the search for suspicious accounts, the system is typically modelled based on the behavioural patterns of the customer or user (Lim et al., 2014). The model is typically fed the following information; previous account history, days passed since account creation and statistical attributes such as the average balance over a month (Lim et al., 2014; Sudjianto et al., 2010). An indication for potential illicit activity arises, when there is a significant deviation from the usual patterns/behaviour for a given account (Lim et al., 2014). On the other hand, transactional-level detection utilises information such as; date of execution, the amount transferred, and location of the sender, in order to classify individual transactions (Lim et al., 2014; Sudjianto et al., 2010). Each setting has its pros and cons, for instance, at an account-level, the system must monitor account by account, which in turn may create an overhead, thus becoming impractical to employ in real-time (Lim et al., 2014). Moreover, even when real-time detection is not a requirement, both settings require aggregating and summarising past information due to the high-dimensional data which may arrive at high volumes and velocity (Sudjianto et al., 2010).

The same settings (account and transactional level detection) and processes (aggregation of previous data) have also been adopted when employing machine learning to detect illicit activity on blockchain networks (Bartoletti et al., 2018; Farrugia et al., 2020; Samanta et al., 2019; Toyoda et al., 2017; Weber et al., 2019). For instance, Farrugia et al. (2020) made use of aggregated information based on the transaction history for particular Ethereum accounts, in order to identify accounts which are deemed as illicit (account-level detection). These attributes included information such as; the total number of transactions executed and minimum/maximum amount of values ever sent/received (Farrugia et al., 2020). Conversely, Weber et al. (2019) utilised aggregated transactional data to identify whether a transaction broadcasted to the Bitcoin network was executed by a person deemed as illicit (holds private keys to the address broadcasting the transaction). Although the target variable was related to the address executing the transaction, the identification of suspicious activity occurred at a transactional-level. The features attributed to a specific transaction, included both single point attributes and aggregated information, for instance; the fee associated with a particular transaction (single-point) and the average amount spent by the list of inputs (aggregated) (Weber et al., 2019).

It is worth noting that the term account-level detection is commonly used interchangeably with customer-level detection; however, sometimes, there is a clear distinction between the two. Customer-level detection may refer to when a detection system is modelled based on the generalisation of both account and transactional level information (Duman and Buyukkaya, 2008). This type of setting may include additional information such as risk scores and the number of accounts associated with a particular customer (Sudjianto et al., 2010). Customer-level detection is not common in cryptocurrencies as it is almost impossible to identify whether multiple addresses belong to the same entity (Lin et al., 2019); however, clustering heuristics have been proposed (Androulaki et al., 2013; Meiklejohn et al., 2013) with the aim of associating multiple addresses to a specific entity/customer. Lin et al. (2019) and Toyoda et al. (2018) adopted similar clustering techniques to classify Bitcoin entities (clustered addresses pointing to one entity/customer) to the respective service (for example mixer, high yielding investment programs and gambling) they operate in, within the network.

2.3.4 | Transactions as a Graph-Structure

Network analysis and graph learning, are important tools in combating financial crime such as money laundering, as these activities are generally conducted between groups of bad actors (such as multiple transactional hops between on-shore and off-shore shell companies to conceal the identity of the source) as described in Section 2.1. A graph or a network consists of nodes (also referred to as vertices) and edges, and it can be both directed (one-way direction) and undirected (two-way direction), as shown in Figure 2.3 (Tarapata et al., 2018). The main techniques employed when fighting financial crime with the use of graph analysis are; analysing links, association rules, pattern recognition, mining frequent patterns, and clustering (Alsuwailem and Saudagar, 2020; Tarapata et al., 2018).

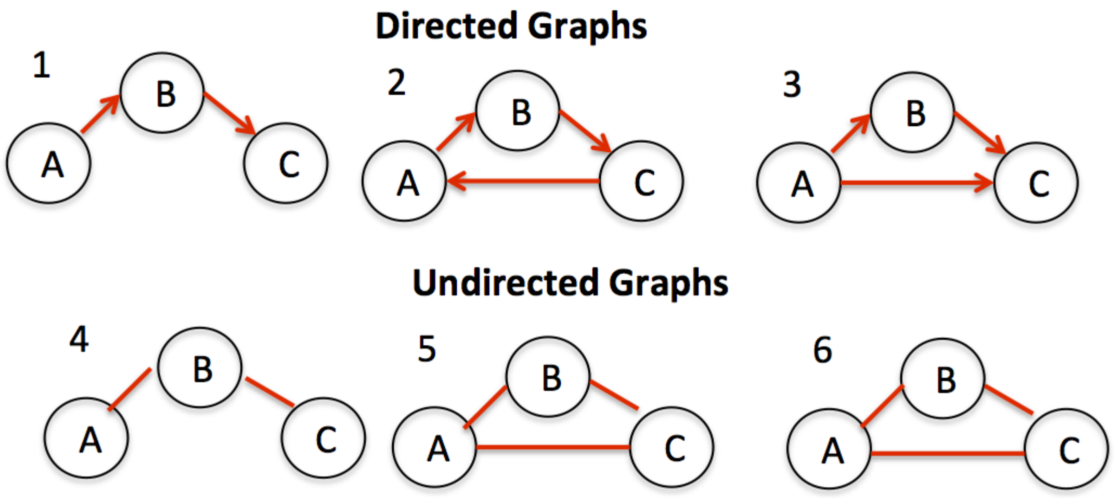


Figure 2.3: Examples of directed and undirected graphs. Circles denote nodes, while the red links denote edges.

A common method used in literature when combating financial crimes on both traditional and cryptocurrency financial systems by utilising network analysis, is the use of transactional graphs (Gaihre et al., 2019; Phetsouvanh et al., 2018; Singh and Best, 2019; Tarapata et al., 2018; Weber et al., 2019). Transaction-graphs are a representation of payment flows (edges) between different users/customers within a system, where edges can contain attributes such as the value transferred when a particular transaction has been executed (Gaihre et al., 2019; Phetsouvanh et al., 2018; Weber et al., 2019).

Fortunately, since cryptocurrency transactions are persisted within a public distributed ledger, transaction-graphs have been utilised to combat financial crime within the cryptocurrency space (Gaihre et al., 2019; Phetsouvanh et al., 2018; Weber et al., 2019). For instance, Gaihre et al. (2019) utilised this structure to deanonymise Bitcoin addresses as a means of fighting crypto crime, while Weber et al. (2019) employed this structure to identify illicit transactions within the same network. Both of these studies made use of a Graph Convolutional Network (GCN) (Kipf and Welling, 2017) to extract graph embeddings in order to transform nodes to a set of vectors. Weber et al. (2019) took it a step further by feeding node embeddings as features to multiple supervised learning algorithms, which proved to be effective when detecting illicit Bitcoin transactions. Additionally, they also investigated this graph learning technique on its own as a classifier, together with two other variants, Skip-GCN and EvolveGCN (handles temporal data) (Pareja et al., 2020; Weber et al., 2019). Pareja et al. (2020) argued that in practice, cryptocurrency transaction graphs could be very large and highly skewed (more skewed than social network graphs), if all transactions were considered, and this can cause issues even for the state-of-the-art graph algorithms.

Representing transactions or accounts as a graph structure have been used to improve interpretability when investigating money laundering or other financial crime rings (Singh and Best, 2019; Weber et al., 2019), which is highly vital for investigators and AML compliance teams. A simple example of a transaction graph within the Bitcoin network is shown in Figure 2.4. In this visual representation, each coloured node denotes an address/account, where an edge represents the payment flows (transactions) between two nodes. The value over the edges represent the total value flowed between two accounts, while the values above the nodes indicate the total number of transactions executed. All nodes in this graph are suspected addresses, with the ones highlighted in a red square considered as *sink accounts*. Sink accounts are addresses used by criminals to aggregate their illicitly-gained funds into one account (Phetsouvanh et al., 2018).

2.3.5 | Skewed Class Distribution

A skewed class distribution (also known as class imbalance) is an intrinsic characteristic in real-world financial crime detection datasets (for example, fraud, scams, money laundering), due to the naturally skewed class distribution in this given domain (Abd Elrahman and Abraham, 2013; Ferreira et al., 2018). This intrinsic imbalance can be recognised by having an estimation of the total overall amount of money/value distributed within the financial system, which can be linked to financial crimes. An analysis carried out by UNODC (2011), estimated that a total of 2.7% of the world's GDP could be linked

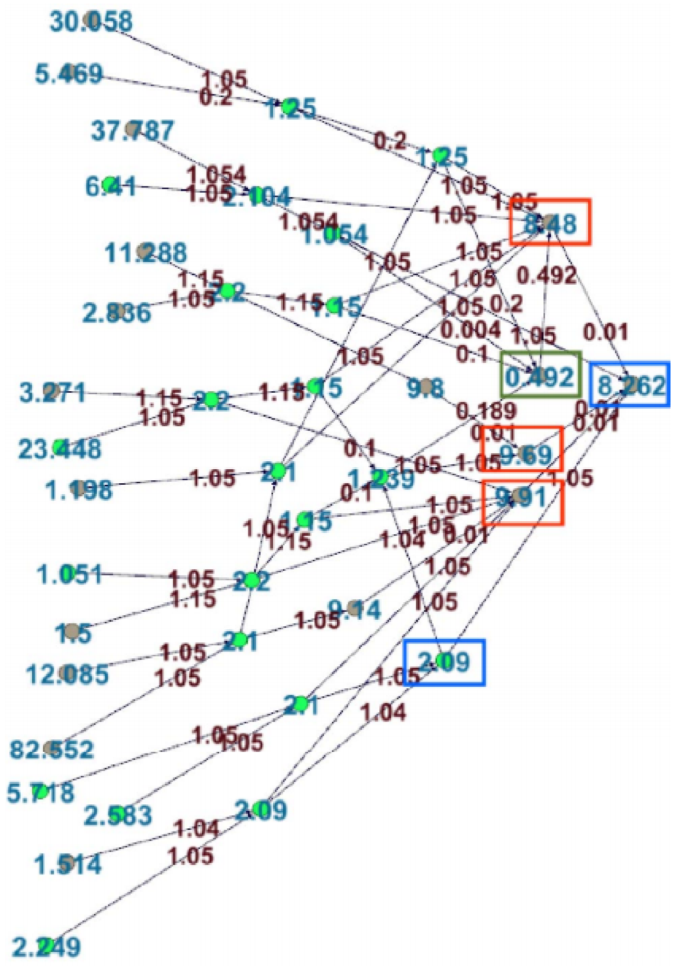


Figure 2.4: A simple example of a transactional-graph within the Bitcoin network (Phetsouvanh et al., 2018).

to money laundering activities. In the cryptocurrency space, it has been approximated that 1% to 2% of the overall activity, can be attributed to criminal activity (Chainalysis, 2020; Elliptic, 2020; Weber et al., 2019).

Since legitimate instances come in more significant numbers than those which indicate illegal activity, the underrepresented samples (minority) are often shrouded by those samples which are overrepresented (majority) (Abd Elrahman and Abraham, 2013; Bartoletti et al., 2018). This issue can cause difficulty to investigators, as detecting illicit activity (minority samples) is usually more of a concern than those deemed as legitimate, and it is like "finding a needle in a haystack". Likewise, class imbalance can cause problems in supervised machine learning as instances of the minority class can be viewed as noise or the algorithm can become biased towards the overrepresented instances (Abd Elrahman and Abraham, 2013; Ali et al., 2015; Leevy et al., 2018). If a skewed class distribution is left unhandled, it can have negative ramifications on the overall performance when identifying the minority samples (Abd Elrahman and Abraham, 2013; Ali et al., 2015; Rout et al., 2018) - which in this study is of utmost importance, as identifying nefarious activity is one of the main interests.

2.3.6 | Non-Stationary Environment

Criminals that partake in money laundering and other financial crimes are continuously exploring innovative techniques towards abusing financial systems, in order to blend in with the norm and avoid being exposed by investigators (van der Veer, 2019). These changes in behavioural patterns employed to stay under the radar, can cause the underlying distribution of the data to change over time (non-stationary). In machine learning, the notion of a shift in the distribution from which a particular model is fitted on, is referred to as *concept drift* (Žliobaitė et al., 2016). In this study concept drift (distribution evolving over time) is defined as follows; (i) a set of instances at a period of time $[0, t]$, can be expressed as $S_{0,t} = \{d_0, ..., d_t\}$ (ii) each instance d_i, can be denoted as $d_i = (X_i, y_i)$ where X_i is a feature vector, and y_i its respective label (iii) the distribution of $S_{0,t}$ can then be denoted as $F_{0,t}(X, y)$ (iv) a change in the underlying distribution, concept drift, happens at $t + 1$ when $F_{0,t}(X, y) \neq F_{t+1,\infty}(X, y)$, which can be expressed as $\exists t : P_t(X, y) \neq P_{t+1}(X, y)$ (Gama et al., 2014; Lu et al., 2019).

The notion of concept drift can be grouped into two, *Real concept drift* and *Viritual concept drift*. The latter refers to when there is a change in the distribution of the feature vector, denoted as changes in $p(X)$ but $p(y|X)$ remain unchanged (Gama et al., 2014). Real concept drift occurs when there is a change in the probability of the target variable y, given a feature vector x, which can be expressed as changes in $p(y|X)$ (Gama et al., 2014). This type of drift can have more of a negative impact on the performance of machine learning algorithms, as it can cause the decision boundaries to change (Gama et al., 2014; Lu et al., 2019). The difference between these two types is illustrated in Figure 2.5.

Additionally, a drift can come in different patterns; Reoccuring, Gradual, Incremental and Abrubt drifts, all of which are shown in Figure 2.6. A reoccurring drift is when previous or new concepts continue to recur over time. A gradual drift occurs gradually, until it dissociates from the previous distribution entirely. An incremental drift, is where the concept changes in a steady progression and an abrupt drift is when a concept changes entirely in an instant (Gama et al., 2014). In an environment where there is a requirement to investigate or analyse evolving data (non-stationary) in real-time, it is

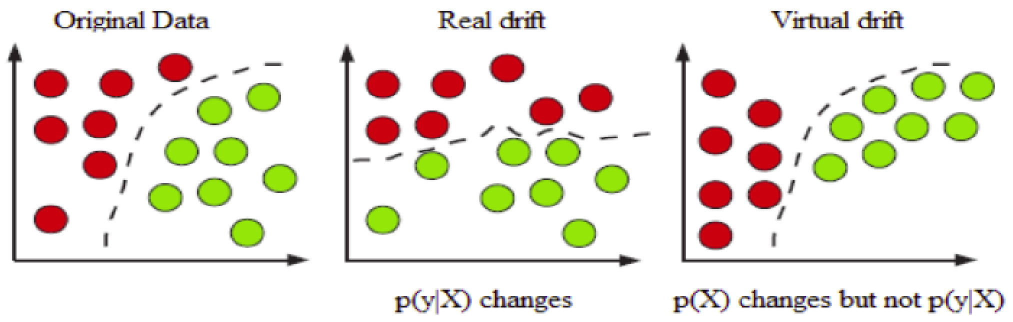


Figure 2.5: The differences between Real and Virtual concept drift, circles indicate samples, while different colours indicate different labels.

essential to handle concept drift, as models built without being aware of these changes can become obsolete over time (Žliobaitė et al., 2016).

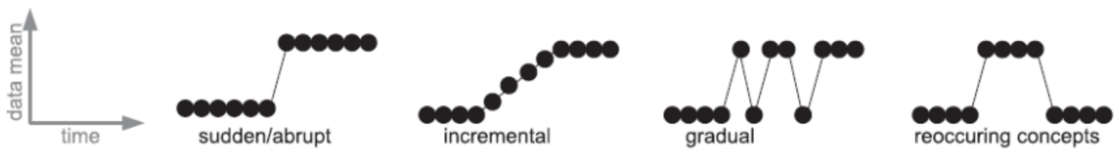


Figure 2.6: Different patterns of concept drift, which include; Reoccuring, Gradual, Incremental and Abrubt drifts (Gama et al., 2014).

2.4 | Ensemble Learning Algorithms

Ensemble learning refers to the process of combining multiple models into one generalised model via various strategic techniques (Dong et al., 2020; Sagi and Rokach, 2018), in order to reduce the overall error by finding a balance between bias and variance (Dong et al., 2020). Understanding the bias-variance trade off is vital, as it dictates the model's overall performance (Dong et al., 2020). Bias describes the difference between the true and the predicted outcome. On the other hand, the variance indicates the deviation from the true functional dependence, or in other words, how sensitive the model is to the training set (Friedman et al., 2001). These two terms can also be viewed as under-fitting (high-bias) and over-fitting (high-variance). The more complex the model is, the more likely it will be prone to over-fitting, while having a simplistic model can result in under-fitting, so a balance between the two is needed to ensure accuracy (Dong et al., 2020; Friedman et al., 2001), as shown in Figure 2.7. In essence, ensemble learning tries to find a balance by utilising and fusing multiple learners (Dong et al., 2020); however, each algorithm has its strategic way of obtaining this balance.

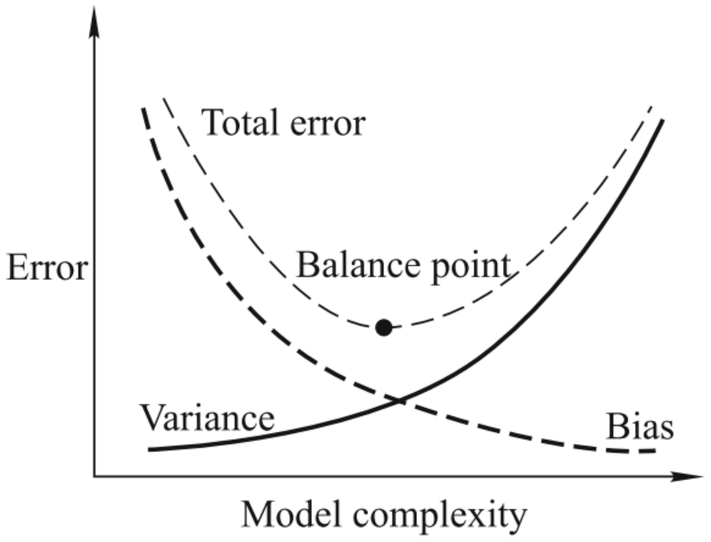


Figure 2.7: Bias-Variance trade-off curve (Dong et al., 2020).

One of the most straightforward techniques to combine multiple classifications into one, is majority voting. In majority voting, each classifier outputs their predicted class, and the final predicted outcome is then taken based on the majority of all predicted classes (Sagi and Rokach, 2018). An illustration of majority voting is shown in Figure 2.8, where seven different classifiers outputted their predicted class. The final prediction for this example is "1" as it got five out of seven votes. There are several other ensemble techniques, and these methods can be categorised in one of the groups shown in Table 2.1. In this table, the 'Fusion Method' refers to how the prediction of multiple base models are combined; for instance, the majority vote method can be viewed as an unweighted fusion method. The dependency refers to how the ensemble is built; (i) in an independent framework, base models are created independently from other base inducers (such as Random Forest (RF)) (ii) in a dependant framework, the next base model is constructed based on the previous inducer (such as Gradient Boosting). Lastly, the final column indicates how the ensemble handles the objective of creating diverse base learners (Sagi and Rokach, 2018). Below a description of the commonly used ensembles (Dong et al., 2020), bootstrap aggregation, gradient boosting and stacking can be found.

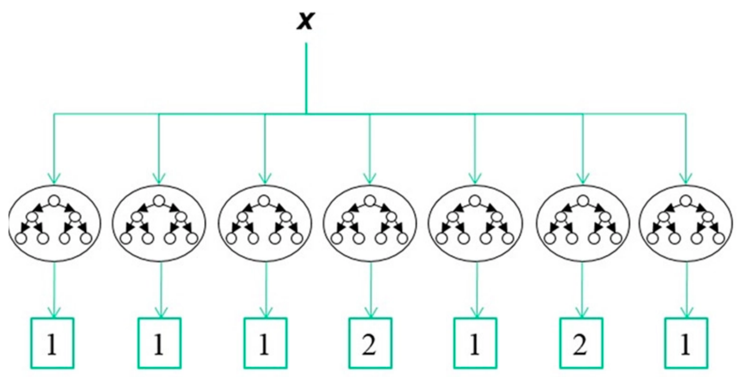


Figure 2.8: An example of majority vote ensemble, the final prediction in this example if "1", as it got five out of seven votes (Sagi and Rokach, 2018).

Method Name	Fusion Method	Dependency	Training Approach
AdaBoost	Weighting	Dependent	Input manipulation
Bagging	Weighting	Independent	Input manipulation
Random forest	Weighting	Independent	Ensemble hybridisation
Random subspace methods	Weighting	Independent	Ensemble hybridisation
Gradient boosting machines	Weighting	Dependent	Output manipulation
Error-correcting output codes	Weighting	Independent	Output manipulation
Rotation forest	Weighting	Independent	Manipulated learning
Extremely randomised trees	Weighting	Independent	Partitioning
Stacking	Meta-Learning	Independent	Manipulated learning

Table 2.1: Different types of ensemble methods (Sagi and Rokach, 2018).

2.4.1 | Bootstrap Aggregation

Bagging, which is also known as Bootstrap Aggregation (Breiman, 1996), builds an ensemble by randomly splitting the training set into random subsamples with replacements (bootstrap samples) and trains homogenous weak base learners using these subsets. The intuition behind bagging, is to reduce the variance of a single unstable learner by aggregating the outputs into a single generalised/aggregated model (Dong et al., 2020; Sagi and Rokach, 2018). The predictions from individual learners is then combined using $sign(\sum_{i=1}^{N} h_i(x))$, where N is the number of base learners in the ensemble and $h_i(x)$ is the predicted outcome of each individual learner. Given that the base learners are independently built, this ensemble can be executed using parallel computation (Sagi and Rokach, 2018).

Although RF (Breiman, 2001) was mentioned as an ensemble method on its own in Table 2.1, it can be considered as an extension of bootstrap aggregation (Sagi and Rokach, 2018). This ensemble is a collection of unpruned decision trees (low bias with high variance), where each tree is created independently. As the name suggests, RF uses randomness to split the data set into bootstrap samples similarly to bagging; however, the base decision tree learners are not only trained on these randomly selected subsets, but a random subset of features are considered to split each node within the individual tree (Breiman, 2001; Sagi and Rokach, 2018). The intuition behind this ensemble is to reduce error by minimising the variance (overfitting) of individual unstable decision trees by combining their outcomes (Dong et al., 2020). Algorithm 1 shows the steps required to create this ensemble in the form of pseudocode. Sagi and Rokach (2018) stated that this model is one of the most popular ensembles in academia, and its popularity can be attributed to; (i) simple but obtains effective performance (ii) can run in a parallel fashion (iii) easy to tune when compared to other ensembles (such as gradient boosting). They backed up their statement by analysing the number of published papers employing RF against other ensembles, for which the reported results are shown in Figure 2.9. Random Forest (RF) also proved its usability in the context of combating crypto-financial crime, by successfully detecting various illicit activities such as; illicit transactions (Baek et al., 2019; Weber et al., 2019), Ponzi-schemes (Bartoletti et al., 2018), and High-Yielding-Investment-Programs (Toyoda et al., 2017).

Algorithm 1 Pseudocode for the Random Forest Algorithm (Sagi and Rokach, 2018).

```
Input: $IDT$(a decision tree inducer),$T$(the number of iterations), $S$(training set),
        $\mu$(the subsample size), $N$(number of attributes used in each node)
Output: $M_t$:$\forall t = 1, ..., T$
for each $t$ in $1, ..., T$ do
    $S_t \leftarrow$ Sample $\mu$ instances from $S$ with replacement.
    Build classifier $M_t$ using $IDT(N)$ on $S_t$
    $t++$
end
```

2.4.2 | Gradient Boosting

Contrary to the previous ensemble, gradient boosting is a technique that builds base learners which are dependant on previously built learners (Sagi and Rokach, 2018).

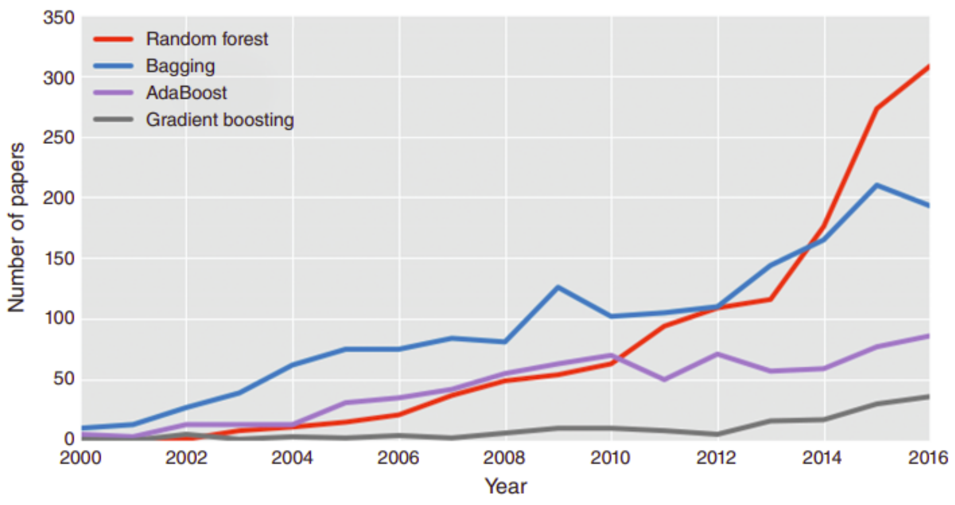


Figure 2.9: The number of published papers per ensemble method, over time (Sagi and Rokach, 2018).

Gradient boosting was introduced by Friedman (2001), and the main objective in this method is to iteratively add homogeneous weak learners in order to reduce the overall loss using a technique similar to gradient descent. Gradient descent reduces error by finding parameters which minimise the cost of a differentiable loss function, whereas in boosting, gradient descents add new learners in a sequential manner (Natekin and Knoll, 2013), as defined in Algorithm 2.

Algorithm 2 Pseudocode for Friedman's Gradient Boost Algorithm (Natekin and Knoll, 2013)

Inputs:

- input data $(x, y)_{i=1}^{N}$
- number of iterations M
- choice of the loss-function $\Psi(y, f)$
- choice of the base-learner model $h(x, \theta)$

Algorithm:

```
1: initialize $\widehat{f_0}$ with a constant
2: for $t = 1$ to $M$ do
3:     compute the negative gradient $g_t(x)$
4:     fit a new base-learner function $h(x, \theta_t)$
5:     find the best gradient descent step-size $\rho_t$:
       $\rho_t = \arg\min_\rho \sum_{i=1}^{N} \Psi\left[y_i, \widehat{f_{t-1}}(x_i) + \rho h(x_i, \theta_t)\right]$
6:     update the function estimate:
       $\widehat{f_t} \leftarrow \widehat{f_{t-1}} + \rho_t h(x, \theta_t)$
7: end for
```
.

The choice of the loss function is arbitrary and should be chosen depending on the problem at hand, as long as the selected function is differentiable (Natekin and Knoll, 2013). This ensemble is typically built by employing decision trees as base learners, and it is worth noting that contrary to RF (builds deep trees), trees are created in a shallow manner. The chosen number of trees is a crucial parameter in gradient boosting, as setting a low number can lead to underfitting, while a high number can lead to overfitting (Sagi and Rokach, 2018). Inspired by bootstrap aggregation (Breiman, 1996), Friedman (2002) introduced the idea of training base learners on random subsamples of the original training set, as a means of reducing the effects of overfitting (Sagi and Rokach, 2018).

More recently, multiple variants of gradient boosting have been developed as further refined/optimised gradient boosting algorithms, in particular, eXtreme Gradient Boosting (XGBoost) (Chen and Guestrin), Light Gradient Boosting Machine (LGBM) (Ke et al., 2017) and CatBoost (Prokhorenkova et al., 2018). XGBoost (Chen and Guestrin) offers a scalable tree-boosting implementation, and it offers a split finding mechanism to handle sparsed data, addresses weighted data using prune/merge techniques and efficiently iterates over possible splits to find the best split. Additionally, it supports distributed frameworks such as Hadoop, and it creates more generalised ensembles by employing regularisation. Light Gradient Boosting Machine (LGBM) (Ke et al., 2017), which is an implementation developed by Microsoft, reduces the memory and computational cost by making use of histogram-based techniques, and it can also be executed in a parallel fashion. CatBoost (Prokhorenkova et al., 2018) uses a weighted sampling variant of stochastic gradient boosting (Friedman, 2002), called Minimal Variance Sampling to find the best split, and this is considered as an improvement to GOSS which was developed for LGBM. This variant of gradient boosting also works with categorical variables without the need for preprocessing. XGBoost does not make use of any weighted sampling; hence the splitting procedure is a bit slower in comparison to LGBM and XGBoost. The application of these algorithms has been explored to detect illicit activities on blockchain networks, for instance, Farrugia et al. (2020) employed XGBoost to detect illicit Ethereum accounts, while Lin et al. (2019) made use of LGBM, to classify Bitcoin addresses to their respective category (for example, gambling and mixer services).

2.4.3 | Stacking

Stacking (Wolpert, 1992), which is the most popular meta-learning technique, is the process of fusing multiple base models by feeding their outputs as an input to another model (referred to as meta-model). Unlike the previously mentioned ensembles, in stacking it is common that the ensemble comprises of heterogeneous learners. This type of learning is useful when base learners can learn different subspaces of the overall problem (Wolpert, 1992).

The notion of stacking is that the meta-model can infer which base-learners (known as level-0 models) are more effective and which are not, by feeding the predicted outcomes (referred to as meta-dataset) outputted by the underlying base-learners as features to the meta-model (known as level-1 models) (Sagi and Rokach, 2018; Wolfinger and yi Tan, 2017). A training set is typically split into two; one used to fit the base learners and the other to compose the meta-dataset. The meta-learner then makes the final predicted outcome (Sagi and Rokach, 2018). It is worth noting that a stacked architecture can also be designed to work with an arbitrary number of levels, but the same concepts still apply (Wolfinger and yi Tan, 2017). Given that the training set is divided, a significant amount of data is needed in order to achieve effective results and employing stacking can come at a computational cost, given the complexity of training multiple models at different levels (Sagi and Rokach, 2018; Wolfinger and yi Tan, 2017). A visual representation for the described process is shown in Figure 2.10, where a simple stacked architecture is presented.

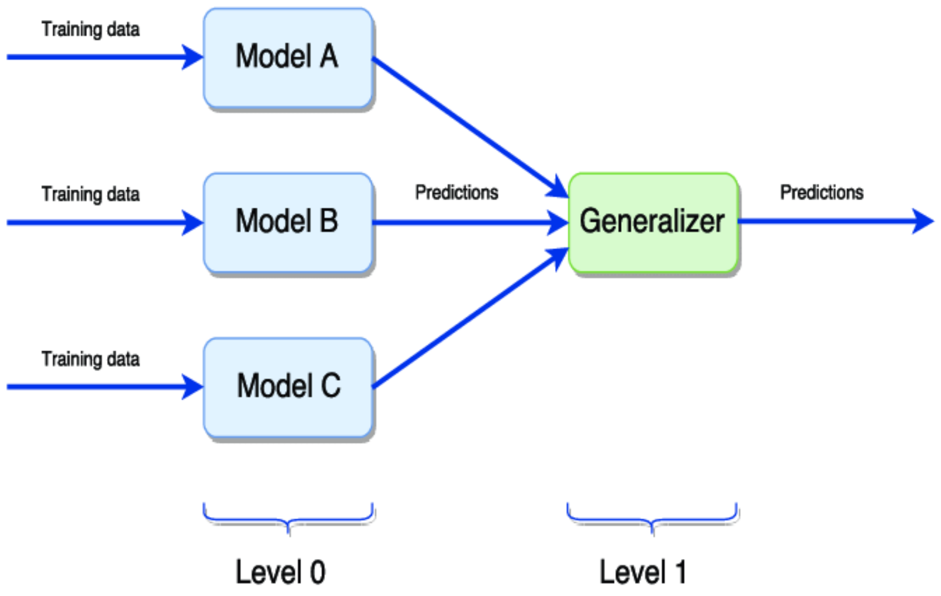


Figure 2.10: A simple example of a stacked architecture (Divina et al., 2018).

Stacking proved to be capable of handling different problems (Sagi and Rokach, 2018), for instance, forecasting the energy consumption to reduce both the economic and environmental footprint (Divina et al., 2018), and in winning various competitions held on Kaggle (Wolfinger and yi Tan, 2017). It was also employed to classify entities on the Bitcoin network to their respective service (for example exchanges, gambling, mixer, and gambling) in order to aid with crypto-crime forensics (Zola et al., 2019).

2.5 | Deep Learning

The application of deep learning has been investigated in the context of this problem and similar domains (Alsuwailem and Saudagar, 2020; Lei et al., 2020; Mubalaike and Adali, 2018; Nilsen, 2019; Pareja et al., 2020; Weber et al., 2019). Deep Learning is a sub-domain of machine learning, where neural networks are constructed in a deep manner, meaning multiple layers of neurons are used to model the data (Johnson and Khoshgoftaar, 2019). One of the simplest examples of a deep learning algorithm is a deep Multi-Layer-Perceptron, where multiple hidden layers are employed to construct a connected feedforward neural net (Johnson and Khoshgoftaar, 2019). A visual example between a Multi-Layer-Perceptron built in a shallow manner versus one built deeply is shown in Figure 2.11.

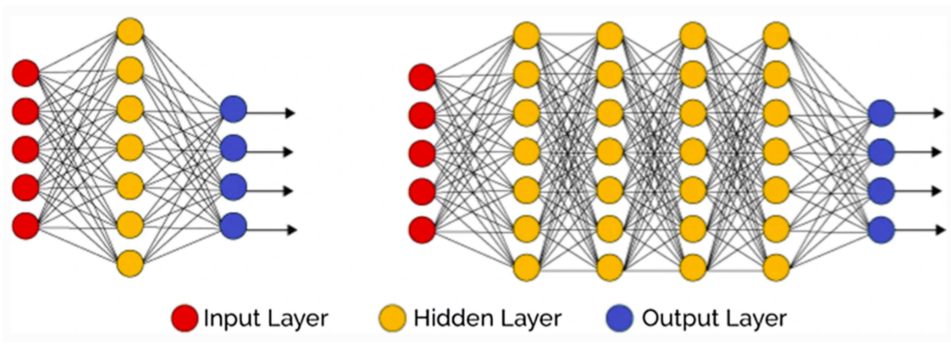


Figure 2.11: A visual representation of an MLP network built in a shallow manner (left) versus one built deeply (right) (Johnson and Khoshgoftaar, 2019).

In traditional machine learning algorithms, the performance achieved is relative to the representation of the data fed to the algorithms; hence the engineering of inputs/features is a crucial step in this type of learning. In complex problems, this phase can be time-consuming; however, in deep learning, this is not much of an issue as a deep learning model can learn through *Representation Learning* (Johnson and Khoshgoftaar, 2019; LeCun et al., 2015). Given the deep/layered architecture of deep learning models, raw input can be transformed into new representations (that is, feature space), enabling the discovery of abstract knowledge at each layer. This process is accomplished through multiple non-linear transformations at each layer, where the level of abstraction increases with each transformation (LeCun et al., 2015). Even though this concept reduces the need for engineering features, and improves the overall performance in complex problems (for example, image recognition), sufficient data must be available in order to extract abstract representations effectively (Johnson and Khoshgoftaar, 2019). The notion of having further layers in a neural net to solve complex problems through abstract representations, ignited the development of other architectures such as Stacked Auto-Encoders and Recurrent Neural Networks (analysing time-series data) (LeCun et al., 2015).

Deep Learning networks presented their effectiveness in various literature in this domain (or similar), some of which are; the use of Graph Convolutional Network (GCN) to detect illicit activities on the Bitcoin network (Pareja et al., 2020; Weber et al., 2019), Generative adversarial networks (GAN) to tackle class imbalance in credit scoring data (Fiore et al., 2019), Stacked Auto-Encoders in conjunction with a Logistic Regression layer (utilising semi-supervised learning) to detect financial fraud (Mubalaike and Adali, 2018), and Long-Short-Term-Memory network to detect Pump-and-Dump schemes on crypto exchanges (Nilsen, 2019). Although these models are useful in complex domains, they are not always a resolution for all machine learning problems, as multi-layered networks require a significant amount of data to be able to converge given the large number of parameters. An investigation/evaluation of whether these learners apply to the target problem must be conducted before utilising the power of deep learning (LeCun et al., 2015).

3

Literature Review

In this section, we will be discussing previous literature investigating various machine learning approaches concerning money laundering detection and other related domains. In the second half of this chapter, a discussion on previous literature carried out to tackle common problems in machine learning independent of the domain, namely, *hyperparameter optimisation*, *class imbalance* and *concept drift* is given, as these issues can be considered as sub-problems in any machine learning solution. The motivation behind this chapter is twofold; to provide a critical analysis of existing literature and to identify any potential gaps.

3.1 | Illicit Activity Detection in Financial Systems

The utilisation of machine learning in financial systems covers a wide range of applications such as; money laundering detection (Alsuwailem and Saudagar, 2020; Jullum et al., 2020; Liang et al., 2019), fraud detection (Bhattacharyya et al., 2011; Fiore et al., 2019; Monamo et al., 2016) and credit assessment (Ferreira et al., 2018; Lei et al., 2020).

One of the most prominent type of models investigated in these applications are Decision Tree (DT) based models. In the context of money laundering, the application of DT models date back to 1995, when Senator et al. (1995) briefly discussed the usefulness of adding DT models to their detection system, however, they stated that the lack of labelled data made it difficult to use these techniques, despite some test results showing the potential of such models. Other research published (Harlev et al., 2018; Jullum et al., 2020; Savage et al., 2017; Weber et al., 2019; Zhang and Trubey, 2019) in subsequent years, claim that tree-based classifiers have great potential in money laundering detection.

Savage et al. (2017) evaluated both SVM and RF in conjunction with network analysis and community detection, to classify whether transactional data provided by the *Australian Transaction Reports and Analysis Centre (AUSTRAC)*, could be classified as being part of money laundering rings. From their empirical analysis, they claimed that RF outperformed SVM in terms of Area Under the ROC curve (AUC), Recall, Precision and F1-Score, however, they remarked that their models could be impacted by other factors which were left unhandled, such as concept drift. Zhang and Trubey (2019) on the other hand, investigated a much broader range of models and also tackled the data imbalance problem using various sampling techniques (over and under-sampling). The dataset employed in their research was provided by a financial institution based in the United States. In their investigation, they tested Bayes Logistic Regression (LR), DT, RF, SVM, and ANN, in order to classify whether alerted money laundering cases were filed as *Suspicious Activity Reports (SARs)*. From their final results, they reported that the best performing models were ANN, Bayes Logistic Regression and RF, ranked from best to worst, when evaluated using the Receiver Operating Characteristic Curve (ROC) and AUC. It was also noted that data-sampling techniques improved the overall performance across all tested models. Jullum et al. (2020) took the approach to apply eXtreme Gradient Boosting (XGBoost) as a means of detecting money laundering at a *transactional-level* by utilising transactional data provided by a Norwegian Bank. This model was specifically selected due to its efficiency, scalability (parallelisation), and its ability to be trained using a GPU (reduces the time for training). They stated that unlike financial fraud detection, where a given dataset typically comprises of instances marked as fraud or legit, in money laundering, the labels are based on suspicious behaviour. This inconvenience is because when a money laundering case is reported to authorities, the reporting institution is rarely ever given feedback as to whether the person was actually charged with wrongdoing. From an empirical evaluation, they proved that XGBoost can efficiently predict whether a transaction should be reported, even outperforming the bank's current system (rule-based with manual inspection). A noteworthy statement from this study was that both non-reported alerts and legitimate transactions which go through different stages of processing (as shown in Figure 3.1), should be utilised in the training phase. If these are left out (which is quite common), it can lead to sub-optimal performance (Jullum et al., 2020).

These ideas can also be employed to detect money laundering activities on Blockchain networks, as the process of laundering money in cryptocurrency is archetypal to the process used in traditional finance as discussed in Section 2.1, however, this environment has its dynamics such as the pseudonymous nature of the network (Adam and Maitland, 2018; Bryans, 2014; Mabunda, 2018) and lack of ground truth (Monamo et al.,

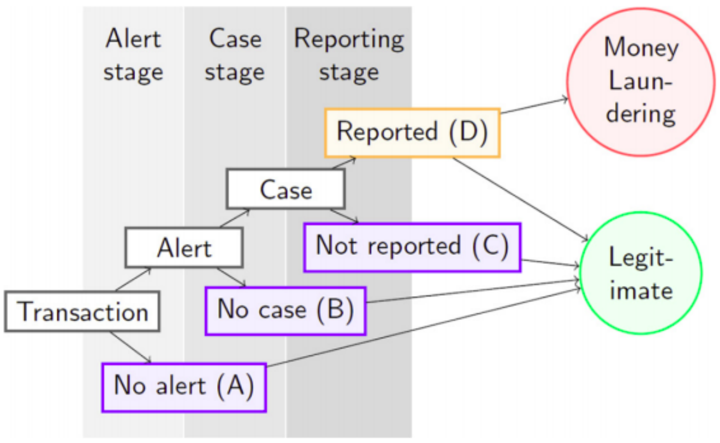


Figure 3.1: The common process flow employed in the industry used to report money laundering (Jullum et al., 2020)

2016).

3.1.1 | Illicit Activity detection in Blockchain Networks

In order to circumvent the lack of unlabelled data which indicate fraudulent Bitcoin accounts, Monamo et al. (2016) investigated the use of *k*-means and *kd*-trees to group atypical Bitcoin accounts based on graph information (i.e. in/out-degree, triangle count) and basic transactional history (i.e. total value sent/received). The top 1% accounts which deviated from local and global neighbourhoods based on the distance to the centroids, were considered fraudulent. Additionally, the newly formed labelled dataset was utilised for training by following classifiers; RF, Maximum-Likelihood LR and Boosted LR, where RF performed the best in terms of Kappa, Recall and Precision. Toyoda et al. (2017) utilised data crawled from online sources (i.e. forums pointing out addresses linked to scams) to detect addresses linked with *"High Yielding Investment Programs (HYIP)"*, a common scam operating on the Bitcoin network. Unlike the previous study (Monamo et al., 2016), their proposed approach focused on transactional patterns (i.e. frequency of transactions) as attributes rather than graph information, to train classifiers. The key idea behind their study was the way they extracted/processed transactional data, where two techniques were proposed - (i) summarise transactional information for each address (ii) summarise transactional information based on owner, which can be deduced using a heuristic approach, where input addresses in a given transaction can be attributed to the same entity/person. RF and XGBoost were then evaluated on this dataset, and it was reported that RF outperformed the other classifier on both schemes (Toyoda et al., 2017).

Extending on these ideas (Toyoda et al., 2017) by employing multi-class classification, Toyoda et al. (2018) showed how RF was able to classify Bitcoin accounts to the respective service they operate in, e.g. gambling and mixer services, with an accuracy score of up to 72%. The ability to adequately classify accounts to their respective service, is essential to combat money laundering, as Fanusie and Robinson (2018) pointed out that money laundering can come from different types of services, with the highest exploited services being; mixer services, dark marketplaces and gambling sites. Bartoletti et al. (2018) set out to detect Bitcoin accounts related to *Ponzi Schemes* using three different classifiers, Bayes Network, Repeated Incremental Pruning to Produce Error Reduction (RIPPER) and RF. Unlike the previous studies (Monamo et al., 2016; Toyoda et al., 2017, 2018), data-sampling approaches and cost-sensitive learning was applied to handle class imbalance. Similarly to Toyoda et al. (2017), the models were evaluated on two different schemes (account and owner based), and from the empirical results, RF proved to be superior. It was also noted that data-sampling approaches overall improved performance across all classifiers, similar to the study conducted by Zhang and Trubey (2019).

Harlev et al. (2018) took a similar approach as Toyoda et al. (2018), since they set out to classify entities holding one or more Bitcoin accounts to their respective type (multi-class), however, these types were more granular, featuring labels such as ransomware, merchant services and hosted wallet services. Additionally, a much broader set of classifiers were evaluated, which included; K-Nearest Neighbours (KNN), RF, Extra Trees, AdaBoost, DT, Bagging Classifier and Gradient Boosting. Synthetic Minority Over-Sampling (SMOTE) was also employed to handle class imbalance, and unlike the previously discussed studies (Bartoletti et al., 2018; Monamo et al., 2016; Toyoda et al., 2017, 2018), Random Search was utilised to tune the hyperparameters. From the reported results, the best performing model was the Gradient Boosting classifier obtaining an F1-Score of 0.75 (Harlev et al., 2018). Contrary to the study conducted by Monamo et al. (2016), Gradient Boosting outperformed RF in this research, however, many factors come into play such as the quality of the data, and one being a binary problem while the other being a multi-classification problem (Harlev et al., 2018). Liang et al. (2019) also targeted a similar multi-classification problem, with four different categories being available, including, miner pool, gambling, general purpose and services (i.e. financial services). The key idea in this research was that the evaluated classifiers were trained on node embeddings extracted from a Bitcoin transaction-graph by utilising a DeepWalk (Perozzi et al., 2014) model. From their evaluation, they showed that feeding node embeddings as features to supervised learning classifiers can be useful in

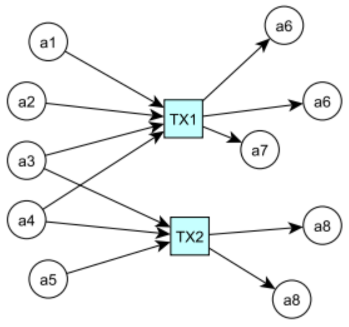


Figure 3.2: Address to Transaction Graph. Circles represent addresses and blue squares represent transactions. The direction of the arrow indicate if an address was in the input list (pointing to transaction) or in the output list (pointing to an address) (Zola et al., 2019).

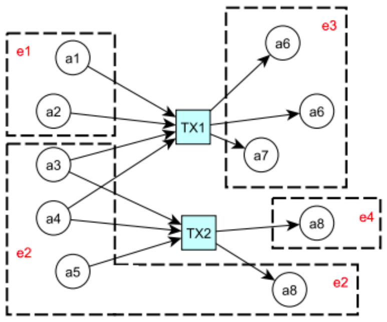


Figure 3.3: Owner to Transaction Graph. Circles represent addresses and blue squares represent transactions. Addresses within a box represent that these accounts are owned by the same entity/person. The direction of the arrow indicate if an address was in the input list (pointing to transaction) or in the output list (pointing to an address) (Zola et al., 2019).

multi-class account type detection. Zola et al. (2019) also made use of graphs to identify Bitcoin addresses to their respective type (i.e. mixer, exchange, gambling), however, two variations of graphs were constructed, one based on addresses and the other based on owners, which are better represented in Figures 3.2 and 3.3. This is a similar idea to the previously mentioned studies (Bartoletti et al., 2018; Toyoda et al., 2017) but in the form of graphs. The core idea behind this study is to extract features from motif graphs and feed them to supervised classifiers, in particular, Gradient Boosting, AdaBoost and RF. A visual representation of the extracted motif data for the owner-to-transaction graph is shown in Figure 3.4. The best performing model in this study was the Gradient Boosting classifier with motif data (Zola et al., 2019).

Gradient Boosting also showed to be useful in the study conducted by Sun Yin et al. (2019), where they set out to de-anonymise Bitcoin accounts. The main objective was to learn the various kinds of transactions in the Bitcoin ecosystem, as a means to aid stakeholders in compliance and regulation (i.e. classify high-risk counterparties which are at a risk of money laundering). "De-anonymising" is a term that refers to categorising Bitcoin accounts to their respective category. The available labels included, personal wallets, stolen bitcoins, ransomware, darknet marketplaces, exchanges, merchant

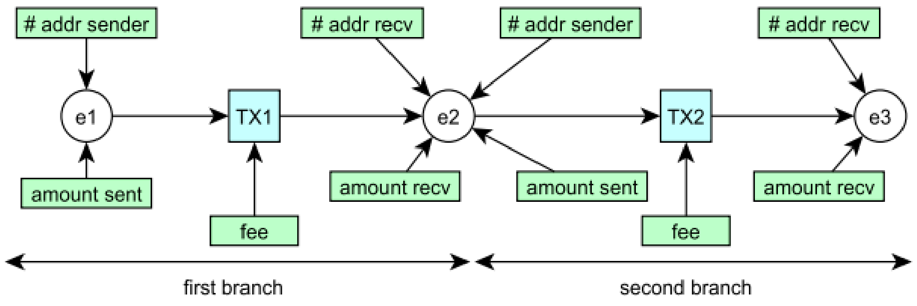


Figure 3.4: Representation of the information/features extracted from the motif graph. The ones marked in green represent features (Zola et al., 2019).

services, hosted wallet services, mining pools, scam, personal wallets, gambling and other (i.e. donation accounts). The dataset was labelled through clustering, similar to the previously mentioned studies (Bartoletti et al., 2018; Toyoda et al., 2017; Zhang and Trubey, 2019), however, clustering was based on information gathered by intelligence (i.e. court documents and data leaks), behavioural clustering and multi-input clustering (input addresses are most likely linked to one entity). A visual representation of clustered addresses is shown in Figure 3.5. An extensive selection of supervised classifiers were evaluated including, KNN, Gradient Boosting, DT, Bagging, Extra Trees, RF and AdaBoost. Hyperparameter optimisation (Random Search) and SMOTE were also employed. From the reported results, Gradient Boosting with default parameters proved to be superior in this multi-classification problem, with an average score (over k-folds) of 80.42% for accuracy and an F1-Score of 79.64%.

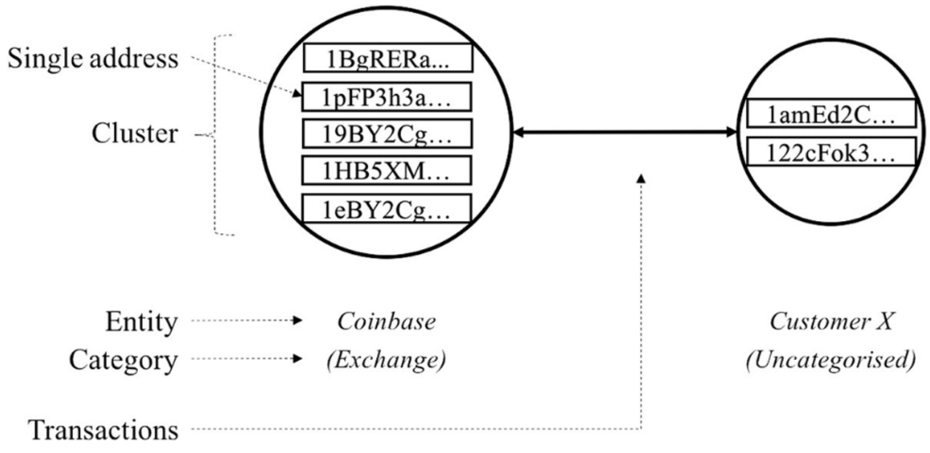


Figure 3.5: A simple example of clustered Bitcoin addresses (Sun Yin et al., 2019).

Tackling a similar problem, Lin et al. (2019) evaluated two other variants of Gradient

Boosting, particularly, XGBoost and LGBM alongside with, LR, Perceptron, SVM, AdaBoost, RF and a Neural Network. However, the labels were not as fine-grained as the dataset utilised by Sun Yin et al. (2019), as it only contained seven different labels. Extending on the ideas set out in previously discussed studies (Toyoda et al., 2017, 2018), they utilised transactional patterns as feature sets, however, they introduced the notion of *Transaction Moments* as a means of capturing temporal information. In simple terms these moments include, mean, variance, skewness and kurtosis of transaction history related to specific accounts, i.e. four moments of distribution for the received and spent transactions. The notion of classifying both individual and clustered addresses which are extracted from the Bitcoin network, was also investigated. From the empirical evidence, the LGBM was the most stable classifier in terms of Micro/Macro F1-Score (87/86 %). It is worth noting that from the feature importance values, they could deduce that *Transaction Moments* contributed to the overall performance.

Contrary, to all the previously mentioned studies in Section 3.1.1 above, where the primary focus was on the Bitcoin Network, Baek et al. (2019) investigated the idea of capturing Ethereum accounts. The main goal underpinning this study is to provide insight into how addresses could be labelled with the utilisation of *k*-means clustering, which also proved to be useful in a study conducted by Monamo et al. (2016). Using these techniques, they clustered data crawled from *etherscan.io* [1] corresponding to wallets used by Binance exchange [2], into seven different clusters. From these clusters it was pointed out that accounts from one of the clusters, contained suspicious transactions, such as transacting anonymously to various accounts on other exchanges, in particular, Bittrex [3]. Additionally, they trained a RF model to classify accounts as suspicious or not, using features based on statistical moments concerning transactions, similar to the study conducted by Lin et al. (2019). The core ideas behind this study were (i) the use of both unsupervised/supervised techniques (ii) engineering features using *Expectation Maximisation* to be fed to a RF model, in order to label accounts (iii) provide insights on how anomalous Ethereum accounts could be pointed out. Moreover, it was shown that some wallets operating on exchanges could have been conducting suspicious activity, in particular on the Binance exchange. Farrugia et al. (2020) also focused on the detection of Ethereum accounts, in particular, identifying whether an account is illicit or not, based on transaction history. Gathering data from Etherscamdb [4] and a local Ethereum client node [5], they formulated a dataset of 2179 illicit (flagged by the community) ac-

[1] Etherscan: `https://etherscan.io/`
[2] Binance Exchange: `www.binance.com/`
[3] Bittrex Exchange: `www.bittrex.com/`
[4] Cryptoscamdb (previously known as Etherscamdb): `https://cryptoscamdb.org/`
[5] Geth Client: `https://github.com/ethereum/go-ethereum`

counts and 2502 legitimate accounts. Unlike the study conducted by Baek et al. (2019), an extensive investigation of various features (i.e. total value sent to smart contracts and total ERC20 tokens sent/received) which can contribute to the overall performance was conducted. With the use of hyperparameter optimisation (Grid Search) and an XGBoost classifier, they successfully classified illicit activity on the Ethereum network at an account-level, as shown in Table 3.1.

No. of Folds	Optimal Depth	Optimal No. of Estimators	Accuracy	F1-Score	AUC	Execution time (s)
3	4	300	0.960	0.957	0.993	62.03
4	4	200	0.960	0.957	0.993	91.13
5	3	250	0.962	0.959	0.994	105.40
10	4	150	0.963	0.960	0.994	230.60

Table 3.1: Results obtained by an XGBoost classifier in Farrugia et al. (2020) study. The table shows the resultant Accuracy, F1-Score, AUC and execution time, corresponding to the selected optimal hyperparameters.

Contrary to the studies discussed so far, in the context of detection on the Blockchain, Lee et al. (2020) explored ways to detect whether individual *transactions* were legal or illegal. Data was collected via online sources [6] [7] and using the transaction's hash, they marked transactions associated with a well-known dark market place SilkRoad (as of today this market has been shut down by law enforcement) [8] as illegal. In total they utilised nine relatively simple features (i.e. the number of input/output addresses, the value of input/output, size and amount) in order to train RF and ANN classifiers. In order to alleviate the problem of class imbalance, various minority to majority ratios were employed, similar to other research (Bhattacharyya et al., 2011; Khoshgoftaar et al., 2007; Roy et al., 2015). Empirical evidence showed that RF was superior in this study, with F1-scores ranging from 0.93 to 0.98, although, it was noted that in practice (live monitoring of transactions) the scores obtained may be be lower, given that there were indications of over-fitting (Lee et al., 2020).

Weber et al. (2019) on the other hand, made use of more sophisticated approaches to detect illicit transactions on the Bitcoin network and made use of a more substantial dataset. The feature set utilised in this study contained graph information in relation to transactions such as (i) local features - number output/inputs, mean of in/out-going

[6] WalletExplorer: `https://www.walletexplorer.com/`

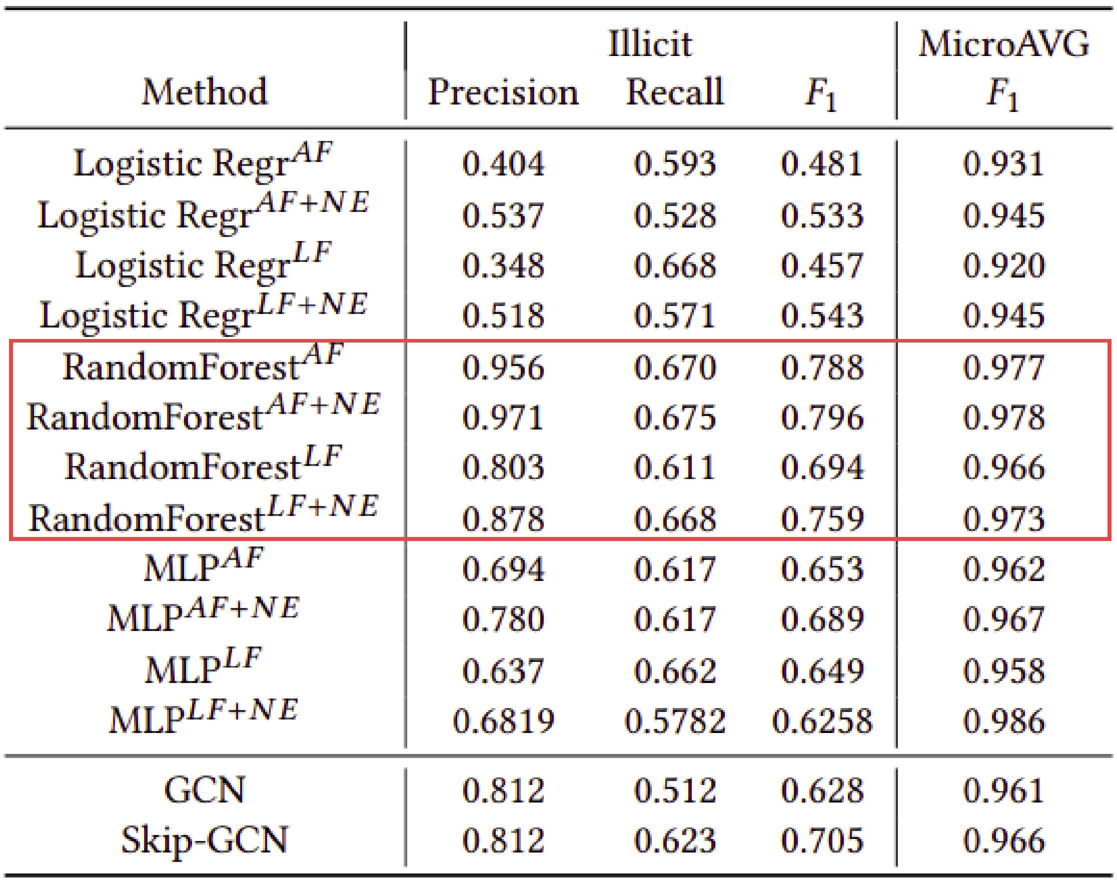
[7] Blockchain Explorer: `https://www.blockchain.com/explorer`

[8] Anonymity is dead and other lessons from the Silk Road trial: `https://www.engadget.com/2015-02-08-silk-road-trial-lessons.html`

transactions corresponding to inputs/outputs (ii) aggregated features - min/max and standard deviation of correlation coefficients of neighbourhood transactions for transaction fees and input/outputs (Weber et al., 2019). The extraction of these attributes was made possible, since transactions were represented as a graph, were vertices denoted transactions and edges denoted the flow of payment. Another level of complexity in this study was that every instance had a temporal component, that is timesteps associated with each transaction. The following models were evaluated, RF, GCN (and multiple variants), LR and Multi-Layered Perceptron. The dataset used for evaluation was split based on timestep, as a means of not breaking the sequence of time. A noteworthy addition, is that they investigated the use of node embeddings extracted from a GCN as input features, for all the tested models (except for GCN). From their findings, they deduced that node embeddings did improve the performances for almost all the evaluated models. Empirical evidence also showed that the RF classifier outperformed the deep learning graph algorithm, GCN, which obtained an F1-Score of 0.796, as shown in Table 3.2.

3.1.2 | Stream Learning on Financial Data

Somasundaram and Reddy (2019) investigated ways to detect fraudulent credit card transactions while also tackling the negative ramifications of class imbalance and concept drift. In order to do so, a new batch incremental adaptive learner coined *'Transaction Window Bagging'* was developed. In essence, this proposed ensemble employs bootstrap aggregation and the utilisation of Cost-Sensitive learning on the underlying base models, to handle class-imbalance (Somasundaram and Reddy, 2019). The incremental learning process works by (i) sampling incoming transactions into bags (giving importance to the minority samples) (ii) training a new Cost-Sensitive Naive Bayes learner and adding it to the ensemble (iii) older models are replaced with new ones (iv) the prediction is made using a weighted vote (more recent having more weight). One of the vital components in this model is that it can detect transactions in an online environment, while also being able to run in parallel (scalable), however, it is only useful in counteracting gradual drifts (Somasundaram and Reddy, 2019). Another recent approach developed to detect fraudulent transactions is the *'FraudMemory'* detection algorithm, proposed by Yang and Xu (2019). Unlike *'Transaction Window Bagging'* (Somasundaram and Reddy, 2019), the key point of this model is to provide interpretability while also improving performance. In order to do so, they fuse user-profiles information and transaction logs into a transactional graph structure, while applying *Gated Recurrent Unit* and memory networks to handle concept drift. They argue that unlike other



Method	Illicit Precision	Illicit Recall	Illicit F_1	MicroAVG F_1
Logistic Regr^{AF}	0.404	0.593	0.481	0.931
Logistic Regr^{AF+NE}	0.537	0.528	0.533	0.945
Logistic Regr^{LF}	0.348	0.668	0.457	0.920
Logistic Regr^{LF+NE}	0.518	0.571	0.543	0.945
RandomForest^{AF}	0.956	0.670	0.788	0.977
RandomForest^{AF+NE}	0.971	0.675	0.796	0.978
RandomForest^{LF}	0.803	0.611	0.694	0.966
RandomForest^{LF+NE}	0.878	0.668	0.759	0.973
MLP^{AF}	0.694	0.617	0.653	0.962
MLP^{AF+NE}	0.780	0.617	0.689	0.967
MLP^{LF}	0.637	0.662	0.649	0.958
MLP^{LF+NE}	0.6819	0.5782	0.6258	0.986
GCN	0.812	0.512	0.628	0.961
Skip-GCN	0.812	0.623	0.705	0.966

Table 3.2: Results reported in Weber et al. (2019)'s study, when investigating the notion of detecting illicit Bitcoin transactions. The best performing model is highlighted in red.

solutions which are typically black-box models, their implementation provides the interpretability which is commonly required among investigators of fraudulent activities (Yang and Xu, 2019).

Weber et al. (2019) took a different approach when trying to handle shifting distributions when detecting illicit transactions on the Bitcoin network. They employed a Graph Learning technique adapted to handle concept drift. Similar to the study conducted by Yang and Xu (2019), they made use of graph information (transactions as nodes and payment flows as edges), however, their approach directly learns from a graph structure, instead of converting graph information as embeddings. They employed *EvolveGCN* (Pareja et al., 2020), which is a variant of *Graph Convolutional Network (GCN)* (Kipf and Welling, 2017) modelled to handle temporal data. The *EvolveGCN* algorithm connects multiple GCNs (created/fitted sequentially with every batch of data) by utilising a *Recurrent Neural Network*, in order to capture shifting dynamics, which is better illustrated in Figure 3.6. From their reported results, they showed that even though the model performed relatively poorly when the environment was stable, once a drift occurred, it was able to adapt more efficiently than other models (Weber et al., 2019). These results are better displayed in Figure 3.7. The key idea in this study, was to learn the problem directly from a graph structure using deep learning graph models, since Bitcoin transactions are in a way modelled as graphs by nature (list of inputs/outputs), however, RF proved to be superior from the overall results, and this could be attributed to the fact that the model performed relatively well when the environment was stable (Weber et al., 2019).

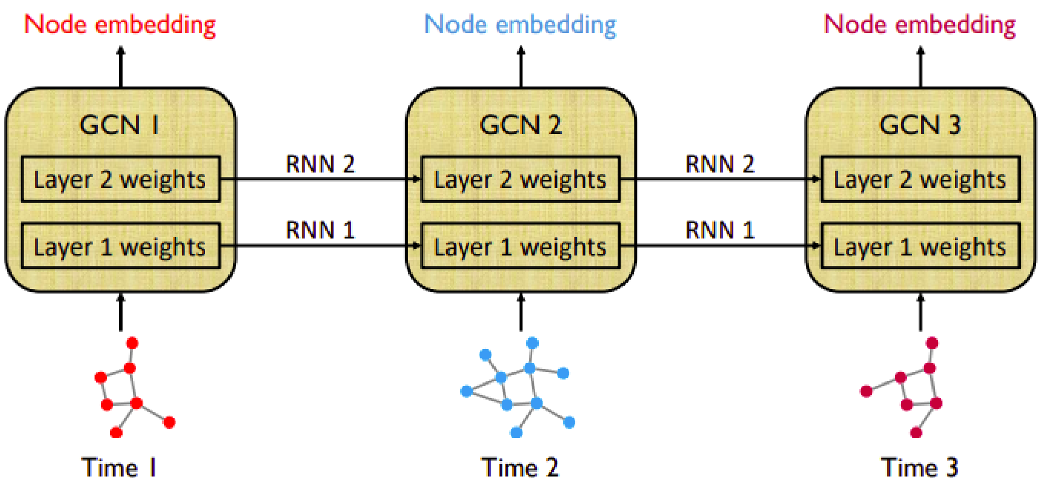


Figure 3.6: Architecture for the EvolveGCN algorithm. With each batch of data a new Graph Convolutional Network (GCN) is trained, and connected via a Recurrent Neural Network in order to capture changing dynamics (Pareja et al., 2020).

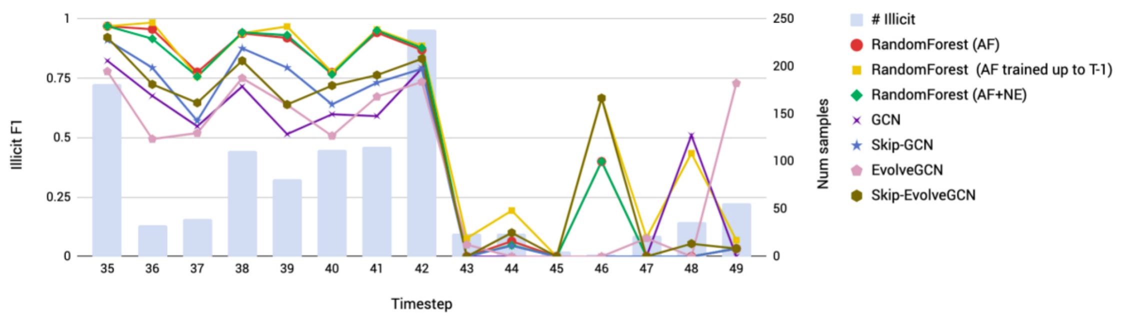


Figure 3.7: F1-Score over different timesteps in Weber et al. (2019) study. The pink line refers to the *EvolveGCN* algorithm.

It is worth noting that previous literature investigating the notion of handling concept drift using adaptive learners, in the context of detecting illicit activity on Blockchain networks, is quite limited.

3.2 | Handling Machine Learning Problems

In this section, previous studies carried out to tackle common problems in machine learning, is given. These problems can be viewed as sub-problems in most machine learning systems, independent of the domain. The literature presented in the coming subsections, focuses on the state-of-the-art machine learning techniques used to address our objectives as defined in Chapter 1. A description linking each section with respect to the objectives, is shown in Table 3.3.

Objective	**Relevant Section**	**Purpose**
Objective. 1	Section 3.2.1	Hyperparameter optimisation is a critical step in finding the optimal parameters which minimise the overall classification error for both the selected state-of-the-art classifiers and the benchmark. This will better aid us in highlighting the most effective model in the context of detecting licit-or-illicit activities on blockchain networks.
Objective 2	Section 3.2.2	Techniques used to handle class imbalance are important in order to improve the detection for the selected classifiers by reducing the negative ramifications of a skewed class distribution.
Objective 3	Section 3.2.3	State-of-the-art techniques used to handle concept drift more effectively are crucial in order to minimise any performance degradation over time, as transaction data is known to be non-stationary.

Table 3.3: Sections focusing on the state-of-the-art machine learning techniques and their corresponding objectives.

3.2.1 | Hyperparameter Optimisation

Hyperparameter Optimisation also referred to as *Hyperparameter Tuning*, is the process of optimising hyperparameters for machine learning models during the training phase, by searching for different hyperparameters which yield the best performance when evaluated on a validation set. Model hyperparameters are specified before the training phase and do not change during the learning phase, such as the number of trees in RF or the learning rate for the Gradient Descent algorithm. These parameters can either be continuous, integers or categorical values and finding an optimal set of hyperparameters can influence performance. The problem of finding the optimal hyperparameters x^* in the domain χ that obtain the highest evaluation score, when assessed on a validation set x, can be formally defined as follows (Bergstra and Bengio, 2012; Hutter et al., 2015).

$$x^* = \arg_{x \in \chi}^{\min} f(x)$$

The definition of the highest evaluation score can be the minimum or the maximum of the objective function $f(x)$, which may correspond to an Error Rate or Accuracy, respectively. The vast majority of hyperparameter optimisation methods can be subgrouped as Manual Search, Grid Search, Random Search, Evolutionary Optimisation and Bayesian Model-Based optimisation (Bergstra and Bengio, 2012; Hutter et al., 2015).

Manual Search is the most naive hyperparameter optimisation method as it works by selecting hyperparameters based on the user's guess, knowledge or intuition, and the larger the search space to search for optimal hyperparameters, the more time-consuming this becomes. Unlike Manual Search, Grid Search works by setting up a grid of hyperparameter values and exhaustively evaluates each combination. The subset of the hyperparameter search space must be manually specified, and due to this reason, this approach requires setting boundaries or discretising continuous values. In some cases, Manual Search is applied in conjunction with Grid Search, to fine-grain the search into the hyperparameter space after honing into an optimal region (Hinton, 2012; Larochelle et al., 2007). This method is relatively straightforward, easy to implement, efficient in low dimensional spaces, and requires little effort to parallelise hyperparameter tuning (Bergstra and Bengio, 2012). However, it is known to suffer from *the curse of dimensionality* (Wilcox, 1961) as the number of combinations grows exponentially with the number of hyperparameters specified (Bergstra and Bengio, 2012).

Bergstra and Bengio (2012) proposed an extension to Grid Search, which they coined Random Search, and the primary intention of their study was to improve efficiency in high-dimensional spaces while preserving the practical advantages of Grid Search. The proposed approach works in the same manner as Grid Search. However, instead of exhaustively enumerating over each combination found in the defined space, it randomly evaluates combinations from the hyperparameter space until reaching the maximum number of iterations set by the user, visually represented in Figure 3.8. They empirically showed that this approach obtained relatively the same performance as Grid Search when evaluated on a 32-dimensional configuration for a deep belief network, with the advantage of explicitly controlling the number of combinations attempted.

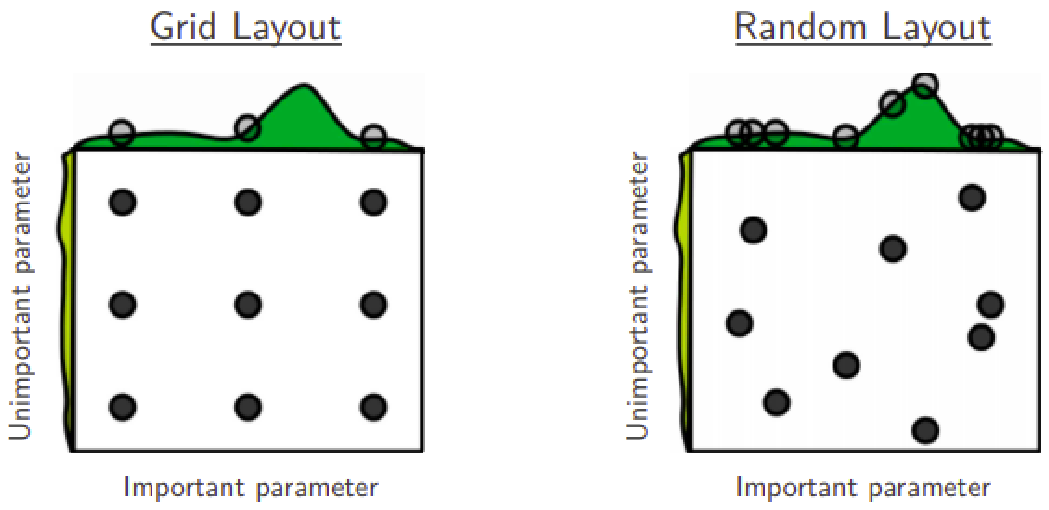


Figure 3.8: (Bergstra and Bengio, 2012).

Another type of optimisation techniques are evolutionary algorithms, and the basic concept is to start with an initial population, which is usually selected at random and then iteratively building more robust populations based on previous generations.This procedure is inspired by the notion of evolution, (natural selection) hence the name *"Evolutionary"* (Bergstra et al., 2011; Chen et al., 2015). One of the most common evolutionary algorithms employed to find optimal hyperparameters is *Genetic Algorithm* (Zames et al., 1981) (Bratton and Kennedy, 2007).

When searching for optimal hyperparameters using Genetic Algorithm, a potential set of parameters is represented as a chromosome. Chromosomes can be specified as binary vectors, string representations or numeric vectors. Once these chromosomes are defined, an iterative process starts, where an initial population of chromosomes undergoes the process of "evolution" to create new generations, with the aim being that each new generation provides better sets of parameters (Schmitt, 2001). This algorithm can be computationally heavy as the *"fitness"* (evaluating a set of specific parameters) for each instance in the population must be computed. Another issue is that this algorithm can converge to a local minimum; however, the use of a mutation factor which injects an element of randomness when a new chromosome is created, can be employed to avoid this. If this mutation factor is set to high, the algorithm will not converge (Schmitt, 2001). Unlike the previously discussed optimisation techniques (Grid and Random Search), Genetic Algorithm employs a heuristic mechanism to find optimal parameters, which in turn can be more efficient in exploring the search space; however, it was pointed out by Chen et al. (2015), that this method suffers from *the curse of dimensionality* (Chen et al., 2015; Xia et al., 2017).

Another approach is Tree-structured Parzen Estimator (TPE), which falls under the structure of *Sequential Model-based Global Optimisation (SMBO)*, which are approaches that sequentially build models in order to choose a new set of potential hyperparameters based on subsequently drawn observations (Bergstra et al., 2011). Unlike the optimisation methods discussed (Random, Grid, Manual Search), this model and other Bayesian approaches use past information to construct a probabilistic model which maps the hyperparameters to a probability on the objective function, $P(score|hyperparameters)$ (Xia et al., 2017). This probabilistic model is referred to as "surrogate", and in TPE a model is constructed by employing the Bayes rule. More specifically this approach models $P(x|y)$ and $p(y)$ instead of $p(y|x)$ (Gaussian Process Approach), with the core idea being to transform the hyperparameters, replacing the distributions of the prior configuration, with non-parametric densities. Xia et al. (2017) showed that this hyperparameter optimisation technique outperformed Grid Search, Manual Search and Random Search when used in conjunction with gradient boosting.

3.2.2 | Handling Class Imbalance

With the increasing demand for applying machine learning to real-world classification problems, such as detecting financial fraud or anomalies, handling a skewed class distribution is a very critical step to take, as it can result in biased predictions (Ali et al., 2015). This classification problem happens when a labelled dataset does not have an equal representation of classes. Underrepresented instances commonly appear in circumstances where the target label is a rare event, or when it is hard to obtain specific samples. If this issue is left unhandled, classifiers may be susceptible to interpret minority instances as noise, or be biased towards the majority class (Datta and Arputharaj, 2018), as illustrated below in Figure 3.9. However, applying techniques to handle skewed class distribution can significantly improve the performance for both deep learning (Johnson and Khoshgoftaar, 2019) and traditional machine learning models (Leevy et al., 2018; Rout et al., 2018). The techniques for handling data imbalance are; Data-Level and Algorithm-Level techniques (Ali et al., 2015).

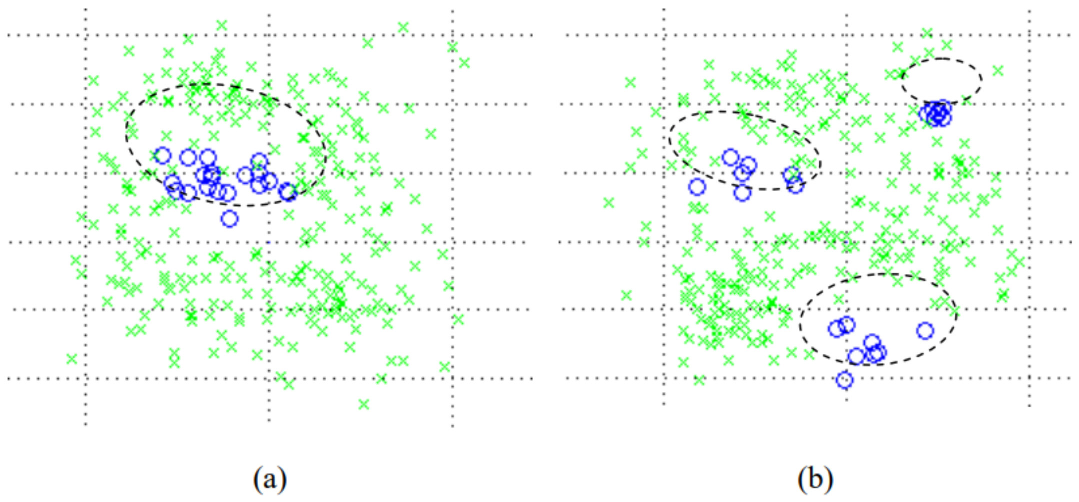


Figure 3.9: Example of class imbalance, the dotted circle(s) represents the decision boundary, the green 'X' represents the majority class instances, while the blue 'O' represents the minority class instances - (a) classes overlapping each other (b) minor disjoint within the decision boundary (Ali et al., 2015)

.

3.2.2.1 | Data-Level Techniques

Data-Level techniques make use of data manipulation, as a means of reducing the repercussions of an unequal class distribution. During the training phase, transformed data is fed into machine learning models to build a solution that is less influenced by class imbalance (Tyagi and Mittal, 2020).

Data Sampling and Feature Selection are subcategories of Data-Level techniques (Leevy et al., 2018). The latter approach involves selecting the essential features, according to the target variable, while also eliminating any irrelevant attributes which could degrade performance. Given this method's characteristic of optimising an evaluation metric through Feature Selection, it has the versatility of enhancing performance on both balanced (Chen et al., 2019; Mafarja et al., 2019) and imbalanced (Maldonado and López, 2018; Pes, 2019; Zhang et al., 2017) datasets when assessed against a proper metric. In a study conducted by Zhang et al. (2017), they employed this procedure to maximise the F1-Score as a means of addressing the problem of imbalanced datasets. Data Sampling methods, on the other hand, try to balance out the skewed class distribution by oversampling the minority instances (Bunkhumpornpat et al., 2009; Chawla et al., 2002; Han et al., 2005) or undersampling the majority instances (Laurikkala, 2001; Tomek, 1976; Wilson, 1972).

Two of the most basic data sampling techniques are Random Over-Sampling (ROS) and Random Under-Sampling (RUS), both of which are non-heuristic approaches which randomly sample instances to balance out the data. ROS randomly creates copies of the minority class to oversample the training set, but this can lead to over-fitting since it duplicates instances of the minority class. On the other hand, RUS randomly removes instances of the majority class until it balances out the class distribution. However, this can lead to losing information which could be invaluable to the trained models (Mohammed et al., 2020). Previous research suggests (Bhattacharyya et al., 2011; Khoshgoftaar et al., 2007; Roy et al., 2015) that instead of randomly removing or adding instances until the target distribution is equal (1:1 ratio), different minority to majority ratios require further investigation and although these techniques are relatively simple, their application should not be disregarded, in place of more sophisticated methods (Rodriguez et al., 2014).

Bartoletti et al. (2018) empirically showed that applying RUS was effective in handling class imbalance, as it was able to improve Recall when classifying Ponzi schemes on the Bitcoin network. In their study, different minority to majority ratios were investigated, including; 1:200, 1:40, 1:20, 1:10 and 1:5. These ratios, were then tested on the following classifiers; RF, Bayes Network and RIPPER and experimental evaluation determined that the 1:5 ratio obtained the highest True Positive Rate. Zhang and Trubey (2019), applied a similar approach when tackling the problem of detecting money laundering at a transactional level. However, they tested different sampling ratios on both ROS and RUS, on a broader selection of classifiers than Bartoletti et al. (2018), with the sampling ratios being from 1:100 up to 40:100 and incremented by 1%. The utilisation of AUC together with Regression Analysis, suggests that both ROS and RUS had positive effects on the models. A recent study conducted by Mohammed et al. (2020), tested both of these Data Sampling methods, on the "Santander Customer Transaction Prediction" [9] imbalanced dataset. From their evaluation, they concluded that ROS was superior to RUS for almost all the classifiers tested. Although ROS obtained higher results when evaluated using various metrics such as Precision, Recall and F1-Score, unlike previous studies (Bartoletti et al., 2018; Roy et al., 2015; Zhang and Trubey, 2019), different sampling ratios were not investigated.

Batista et al. (2005), states that utilising heuristics reduces the shortcomings associated with the use of non-heuristic Data Sampling methods. A heuristic approach to

[9] `https://www.kaggle.com/c/santander-customer-transaction-prediction`

oversample the minority class is Synthetic Minority Over-Sampling (SMOTE), pseudocode for which is shown in Algorithm 3. This technique synthesises instances by first randomly selecting a minority instance, and then randomly selecting one of its k nearest neighbours. A new synthetic minority instance arises after computing the Euclidean distance between the two and multiplying the resulting vector by a random value between 0 and 1 (Chawla et al., 2002). Despite this technique's wide use in existing literature (Harlev et al., 2018; Sayed et al., 2019; Yong Sun and Feng Liu, 2016), a known limitation is that it can create noise when the classes overlap, as it does not consider the majority class (Bunkhumpornpat et al., 2009) when generating synthetic instances.

Algorithm 3 Pseudocode for SMOTE (Chawla et al., 2002)

```
Input:
    $D_{minority}$, Minority Instances
    $N_{percentage}$, Amount of SMOTE
    $k$, $k$-nearest neighbours
Output:
    $D_{smoted}$, Synthetic Instances
 1: function SMOTE($D_{minority}$, $N_{percentage}$, $k$)        ▷ $k$ set to 5 in our implementation
 2:     $D_{smoted} \leftarrow []$
 3:     for $i \leftarrow 1$ to $nrow(D_{minority})$ do
 4:         $nn \leftarrow kNN(D_i, D_{minority}, k)$
 5:         $N_i \leftarrow \lfloor N_{percent}/100 \rfloor$
 6:         while $N_i \neq 0$ do
 7:             $neighbour \leftarrow selectRandom(nn)$        ▷ With Uniform Probability
 8:             $gap \leftarrow rangeRandom(0, 1)$
 9:             $diff \leftarrow neighbour - D_i$
10:             $synth \leftarrow D_i + gap * diff$
11:             $D_{smoted} \leftarrow append(D_{smoted}, synth)$
12:             $N_i \leftarrow N_i - 1$
13:         end while
14:     end for
15:     return $\leftarrow D_{smoted}$
16: end function
```

Therefore, motivated by this drawback, multiple extensions of SMOTE have been developed throughout the years, such as Borderline-SMOTE (Han et al., 2005), which only considers minority samples near the borderline, when generating synthetic minority samples thus reducing overlapping classes (as shown in Figure 3.10). Alternatively, Safe-Level-SMOTE (Bunkhumpornpat et al., 2009), assigns safe levels to each minority instance by making use of its nearest minority neighbours. The intention is to synthesis more minority instances around larger safe levels. Another extension is Adaptive Synthetic Sampling (ADASYN), which makes use of a weighted distribution over instances belonging to the minority samples, as a means of assessing and addressing the learning difficulty. The intention is to shift the decision boundary towards the harder to learn instances, by creating more of them when compared to easier to learn instances, which in turn reduces the bias injected by the skewed class distribution (Haibo He et al., 2008).

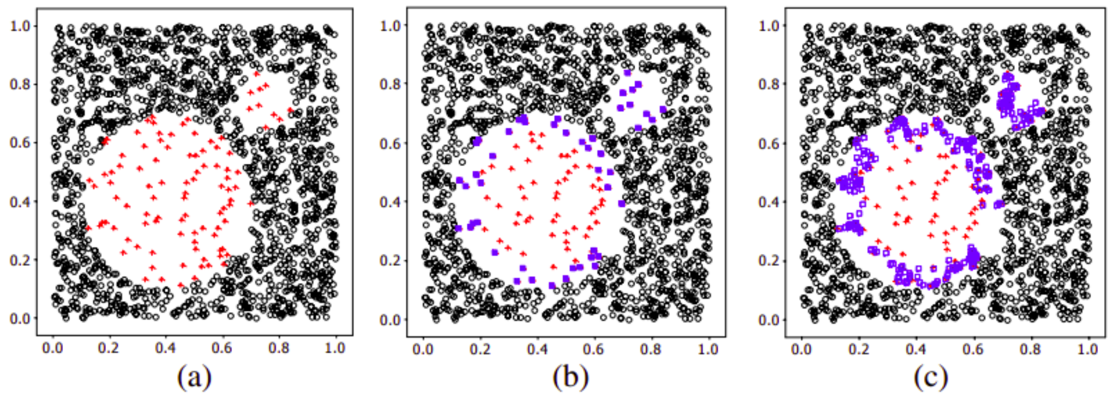


Figure 3.10: (a) Shows the original dataset prior to oversampling (b) Shows minority instances which are around the borderline, displayed as solid squares (c) Shows the synthetic minority observations, displayed as hollow squares (Han et al., 2005)

An undersampling heuristic approach used to handle imbalanced datasets, is the Condensed Nearest Neighbors (CNN) rule (Hart, 1968), which was developed to decrease space complexity for the KNN algorithm (Cover and Hart, 1967) but also showed to be effective in handling data imbalance. The idea behind this rule is to incrementally add majority instances which are close to the decision boundary, which works as follows; construct a subset containing all of the minority instances, if 1-Nearest Neighbour correctly classifies the majority instance, it becomes part of the initial subset, otherwise it is discarded. A criticism of the CNN rule is that it selects samples at random, which could result in retaining internal and ambiguous instances instead of boundary instances (Tomek, 1976). Another heuristic-based undersampling approach is the Near Miss (NM) undersampling. Mani and Zhang (2003), proposed different methods for selecting majority instances in this approach, such as; selection based on the minimum average distance to the three closest minority instances, and selecting instances with the minimum distance to each minority instance. The objective of these methods is to keep instances that are close to the decision boundary.

Both the NM and the CNN rule are known to be "select to keep" approaches, as they try to find majority instances to be kept in the dataset. In contrast, a "select to delete" approach, picks instances to delete from the majority class, such as Tomek Links (TL) (Tomek, 1976) which is an extension of the CNN rule. Tomek (1976), defined a link as a pair of instances that are Nearest Neighbours and belong to opposite classes, as can be seen in Figure 3.11. These links are either boundary or noisy instances which often result in misclassification, and so, by removing the majority instances found within these links, the dataset would be less ambiguous, and the downfalls of the CNN rule improved. An alternative approach to removing ambiguous and noisy data is the Edited Nearest Neighbors (ENN) rule (Wilson, 1972). This rule computes the three Nearest Neighbours of each instance denoted as *a*. The rule removes *a*, if it forms part of the majority class, and its three Nearest Neighbours misclassify this instance. However, the rule will remove the majority instances among *a*'s neighbourhood, if its three Nearest Neighbours misclassifies it and *a* forms part of the minority class. The final sampled dataset will not be equally balanced (1:1 ratio), since the concept behind both ENN and TL is to remove noisy and ambiguous instances.

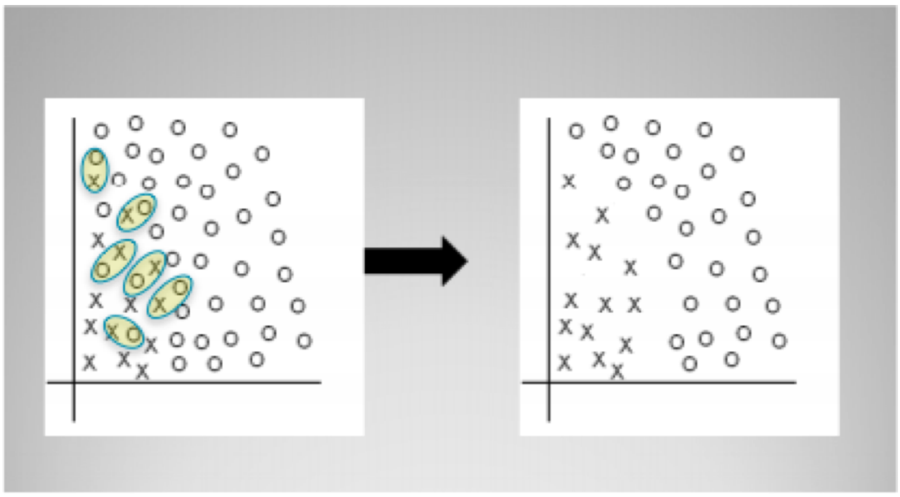


Figure 3.11: The diagram on the left shows pairs of instances which are nearest neighbours and belong to opposite classes. These are considered to be Tomek Links (highlighted in green). The one on the right shows the transformed dataset when these links are removed, resulting into a less ambiguous dataset (Fawcett, 2016).

There are also techniques which combine both the "select to keep" and "select to delete" approaches such as; One-Sided Selection (OSS) (Kubat and Matwin, 1997) and NCL (Laurikkala, 2001). The former combines TL to remove noisy and borderline majority observations, followed by CNN to remove redundant majority instances that are far from the decision border. On the other hand, NCL combines CNN to remove redundant observations and ENN to remove ambiguous or noisy instances. Laurikkala (2001), stated that the quality of the results is not always dependant on the size of the classes but rather, on the quality (less ambiguous) of the observations. Due to this, NCL primarily focuses on "cleaning" the majority samples as opposed to OSS.

Even though these undersampling approaches make use of heuristics to remove majority instances in a more informed way when compared to RUS, Gu et al. (2007) pointed out that the main drawback for these undersampling techniques is that the effectiveness of their performance lies on the distance function defined. Moreover, it is not always a trivial task to find the most effective distance function for a particular domain (Aggarwal, 2003).

A study conducted by Tyagi and Mittal (2020), evaluated several oversampling and undersampling techniques, on different classifiers using various imbalanced real-world datasets. Through experimentation, they concluded that ADASYN and NCL were the best performing oversampling and undersampling approaches, respectively. Furthermore, NCL obtained better overall results when evaluated using measures such as balanced Accuracy, Recall and Precision. Contrary to the traditional Data Sampling approaches evaluated by Tyagi and Mittal (2020), in recent years the use of Deep Learning models such as Generative Adversarial Networks (GAN) are also being explored to oversample imbalanced financial data (Fiore et al., 2019; Lei et al., 2020).

Previous literature also investigated the notion of combining oversampling and undersampling techniques. Yong Sun and Feng Liu (2016) combined SMOTE and NCL on an imbalanced network intrusion detection dataset, to overcome the impact of boundary/noisy data when oversampling with SMOTE. In their experiments, they made use of different classifiers to test their proposed approach, and from their results, SMOTE-NCL obtained higher performance, particularly improving AUC, when compared to SMOTE.

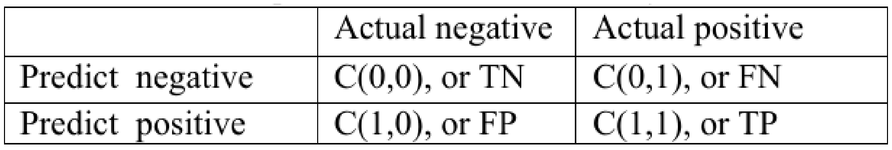
Another approach which utilised both NCL and SMOTE was proposed by Junsomboon and Phienthrakul (2017), as a means of reducing the effects of imbalance in medical diagnosis data. In contrast to the study conducted by Yong Sun and Feng Liu (2016), rather than removing the outliers in the majority observations after SMOTE was applied, they proposed the removal of these instances before oversampling. The primary focus of this study was to improve the Recall measure, and from the results obtained, they concluded that utilising NCL-SMOTE can obtain an overall higher recall score when compared to SMOTE.

Ferreira et al. (2018), investigated a much broader range of Data Sampling techniques when evaluating imbalanced financial datasets, such as; RUS, NM, TL and Cluster Centroid as undersampling approaches, and SMOTE and ADASYN for oversampling approaches. SMOTE, in conjunction with TL, was also evaluated, which is a combination of the two. Overall, the performance of these sampling approaches were fairly similar when evaluated on various classifiers, when measured using different metrics such as AUC and Specificity. However, from the results, it is shown that different classifiers respond differently for each sampling technique and the combination of LR and ADASYN, produced the best combination. They also argued that given the intrinsic imbalance of financial data, oversampling could contribute too little or not at all to performance. They stated that the minority samples usually tend to be represented in a dense region in a large feature space, and so, synthetic data arises in this region, resulting in no new patterns for the classifier to discover. Moreover, even though the performance in detecting the minority class increased, the results in detecting the majority class degraded. Other researchers also noted this trade-off between the performance of the minority and majority class, when applying Data-Level techniques to mitigate the impact of class imbalance (Bartoletti et al., 2018; Pan et al., 2020).

3.2.2.2 | Algorithm-Level Techniques

In contrast to the previous techniques, the algorithm oriented approach handles the data imbalance issue by directly learning the imbalanced distribution, rather than sampling the dataset. Cost-sensitive learning and hybrid/ensemble methods are two sub-groups of Algorithm-based approaches (Ali et al., 2015).

The concept behind Cost-Sensitive Learning techniques is defined as follows; define a Cost Matrix C, where $C(i,j)$ denotes the cost of misclassifying a predicted class i, from its actual class j, as shown in Figure 3.12. By imposing a higher cost when misclassifying minority instances, the emphasis shifts towards them, and the final objective is then to minimise this cost which inherently increases the performance towards the minority (Ling and Sheng, 2010). Bartoletti et al. (2018) noted that employing Cost-Sensitive Learning with a RF model was most effective in balancing out Recall and Specificity in comparison to RUS and other models. However, other researchers argue that this technique has its own set of limitations; the cost of misclassification is hard to capture (Galar et al., 2012; Weiss et al., 2007), there may be a need for domain experts to capture this cost (Ling and Sheng, 2010) and potential overfitting (Weiss, 2004). The technique may also obtain the same performance as oversampling, though it requires an overhead to find an optimal cost value (López et al., 2012).



	Actual negative	Actual positive
Predict negative	C(0,0), or TN	C(0,1), or FN
Predict positive	C(1,0), or FP	C(1,1), or TP

Figure 3.12: An example of a cost matrix for a binary classification problem (Ling and Sheng, 2010).

Another option for Algorithm-Level techniques is hybrid/ensemble methods. Ensemble learning refers to training several classifiers during the training phase to produce a final classification based on their aggregated evaluation. Hybrid/Ensemble refers to a combination of the previously mentioned techniques and ensemble learning. Chen et al. (2004) proposed two variants of RF; Weighted-RF which utilises Cost-Sensitive Learning by placing a more expensive cost when misclassifying the minority instances, and Balanced-RF, which is a combination of RF and undersampling. A recent study conducted by Lahmiri et al. (2020), empirically showed the effectiveness of hybrid/ensemble methods to predict financial risk, in particular; RUSBoost (Seiffert et al., 2010) which is a combination of RUS and AdaBoost (Freund and Schapire, 1997a). RUSBoost outperformed previous literature, making use of the same datasets in terms of Accuracy. However, it is worth noting that Accuracy is not an effective measure to capture the performance on a skewed class distribution (Ali et al., 2015). The independency of the base classifier in hybrid/ensemble techniques produces a more adaptable approach than Cost-Sensitive Learning. However, to ensure performance when constructing ensembles, one must innovate diverse classifiers, while also, conserving their stability within the training data (Ali et al., 2015). Some researchers argue that the concept of diversity in ensembles for classification problems, is still obscure (Wang and Yao, 2009). Galar et al. (2012) pointed out that understanding error diversity is a challenging task, and the complexity issue grows higher when adding more classifiers.

To alleviate the drawbacks of Data-Sampling, Feature Selection, Cost-Sensitive Learning and tuning traditional classifiers, researchers also suggested the hybridisation of utilising one or more of these techniques (Ali et al., 2015; Leevy et al., 2018). From the reviewed literature it is evident that handling data imbalance can improve the performance of machine learning models for both financial (Ferreira et al., 2018; Fiore et al., 2019; Mohammed et al., 2020; Tyagi and Mittal, 2020; Zhang and Trubey, 2019) and cryptocurrency (Bartoletti et al., 2018; Harlev et al., 2018; Liang et al., 2019) imbalanced datasets. However, as noted by Leevy et al. (2018), various techniques should be employed when dealing with class imbalance in a given domain, as there is no general best approach.

Ultimately, when class imbalance is present in a given dataset, it is incredibly crucial for the evaluation phase to make use of performance metrics which capture its skewed class distribution. These measures may include the confusion matrix, and its derivations such as; Recall, Precision and F1-Score (Ali et al., 2015).

3.2.3 | Handling Non-Stationary Data Streams

Class imbalance is not the only intrinsic property of financial data that deteriorates performance, as most of the time, this data can also suffer from concept drift (Somasundaram and Reddy, 2019). As described in Section 2.3.6, concept drift refers to the notion of a shift in the distribution from which a particular model is trained on. This statement is more of an issue when a model is constructed statically, meaning that it was trained in a traditional batch learning setting. Without any interference, a model trained in a batch environment will have a hard time predicting the outcome when concept drift occurs, since it was fitted on a different concept/problem (Montiel et al., 2020). As most financial data is in some way or another a temporal component, that is transactions are timestamped on the blockchain (Weber et al., 2019; Zola et al., 2019), it may be susceptible to concept drift and one way to handle this issue is to build adaptive learners in a stream environment (data is streamed over time) (Montiel et al., 2020).

3.2.3.1 | Adaptive Learners

One of the most common technique employed to counteract concept drift is the use of ensemble learning (Sagi and Rokach, 2018). An early adaptation of ensemble learning utilised to handle streamed non-stationary data, was proposed by Street and Kim (2001). A variation of bootstrap aggregating (Breiman, 1996), the *Streaming Ensemble Algorithm*, comprises of a fixed number of base models, and it can incrementally learn from batches of data. This algorithm is known to be a *passive-approach* learner, as it assumes that drift is always present, so it continually updates the model as fresh incoming data is pushed through the stream (Elwell and Polikar, 2011). Another variant of bootstrap aggregation adapted to handle evolving data-streams is *Online Bagging* (Oza, 2005). Akin to the method proposed by Street and Kim (2001), it too is a passive learner that creates M base models fitted on N batches of data, and predicts unseen samples using an unweighted voting of the underlying M base learners, however, instances picked for training are selected with replacement. Since the data is streamed and $N \to \infty$, they proposed to simulate the replacement strategy by fitting the underlying base learner using K copies of the original batch drawn from a *Poisson(1)* distribution, as shown in Algorithm 4.

An extension of Online Bagging (Oza, 2005), was proposed by Bifet et al. (2010), coined as *Leveraging Bagging*. The notion behind their proposed solution was to apply an element of randomness to diversify and improve the overall performance (accuracy) of the underlying ensemble. Inspired by the use of randomisation in Random Forest (RF) (Breiman, 2001), they proposed to make use of randomisation to attribute weights to the samples in the input stream. Additionally, randomisation at the output was also added

Algorithm 4 Pseudocode for Online Bagging Algorithm (Oza, 2005).

```
1: Initialize base models $h_m$ for all $m \in \{1, 2, ..., M\}$
2: for all training examples do
3:     for $m = 1, 2, ..., M$ do
4:         Set $w = Poisson(1)$
5:         Update $h_m$ with the current example with weight $w$
6: anytime output:
7: return hypothesis: $h_{fin}(x) = \arg\max_{y \in Y} \sum_{t=1}^{T} I(h_t(x) = y)$
```

by utilising output codes (boosts multi-classification) (Schapire, 1997) and they state, that this will reduce the correlation between base-learners in turn, promoting diversity. This model uses Adaptive Windowing (ADWIN) algorithm (Bifet and Gavalda, 2007) in order to detect drifts, and once a change is detected, a new model replaces the worst base-model, as shown in Algorithm 5 (Lines 7-8). From their empirical evaluation, they showed that these proposed improvements increase the overall accuracy when compared against Online Bagging (Bifet et al., 2010), however, this increase in performance comes at a cost, as this model is more expensive in terms of resources (Bifet et al., 2010; Read et al., 2012).

Algorithm 5 Pseudocode for Online Leveraging Bagging Algorithm (Bifet et al., 2010).

```
1: Initialize base models $h_m$ for all $m \in \{1, 2, ..., M\}$
2: Compute coloring $\mu_m(y)$
3: for all training examples $(x, y)$ do
4:     for $m = 1, 2, ..., M$ do
5:         Set $w = Poisson(\lambda)$
6:         Update $h_m$ with the current example with weight $w$ and class $\mu_m(y)$
7: if ADWIN detects change in error of one of the classifiers then
8:     Replace classifier with higher error with a new one
9: anytime output:
10: return hypothesis: $h_{fin}(x) = \arg\max_{y \in Y} \sum_{t=1}^{T} I(h_t(x) = \mu_t(y))$
```

Similarly to the subsequent algorithms (Bifet et al., 2010; Oza, 2005; Street and Kim, 2001), *Accuracy Weighted Ensemble* (Wang et al., 2003) produces new base learners for each batch or time-step. One key difference in this algorithm is that it employs Mean Squared Error to attribute weights to the underlying base models. This procedure is better displayed in the following equations.

$$MSE_i = \frac{1}{|S_n|} = \sum_{(x,c) \in S_n} (1 - f_c^i(x))^2$$

$$MSE_r = \sum_c p(c)(1 - p(c))^2$$

MSE_i denotes the error for a specific base-learner, while the MSI_r is the error of a random model, where $p(c)$ is the probability of selecting a class c at random. Any classifiers with an error $MSE_i \geq MSE_r$ are disregarded. The weight w_i for a specific classifier c_i is computed by subtracting the two, denoted as $w_i = MSE_r - MSE_i$ (Wang et al., 2003). It is stated that this algorithm is efficient at handling reoccurring drifts, as previously added base-learners could become important again, which can be indicated by the weighted procedure discussed. The final predicted outcome in this approach is taken based on k best base-learners, weighted on accuracy (Wang et al., 2003).

Another ensemble to handle shifting distributions is the *Ensemble of Restricted Hoeffding Trees* (Bifet et al., 2012). This adaptive learner comprises of Hoeffding Trees (Domingos and Hulten, 2000) as base learners, that are trained on small random subsets of features. The predicted outcomes (probabilities) of each base tree is fused by employing stacking (Wolpert, 1992) and using a simple perceptron as a meta-learner. ADWIN is used to monitor the performance of each tree, and when there is a significant degradation in performance in one of the trees, that particular tree would be reset. Additionally, the coefficients in the meta-learner (perceptron) associated with that particular tree are reset to zero. Unlike the previously mentioned studies (Bifet et al., 2010; Oza, 2005; Street and Kim, 2001), this model combines the output of the underlying ensemble through another model (meta-learner) rather than using some form of voting. They noted that their proposed adaptive stacked ensemble improved performance of adaptive bagged ensembles (Bifet et al., 2012).

A more advanced concept was proposed by Losing et al. (2016), coined the *Self Adjusting Memory* algorithm, which employs the idea of short and long term memory. Their study aimed to develop an adaptive learner that could handle different patterns of concept drift (described in Section 2.3.6) using memory-models and building an ensemble of learners based on previous and current concepts. They argue that models built only for specific patterns of concept drift can fail or perform sub-optimally as they disregard previous information. This is further amplified in the case of recurring drift. The method utilises two types of memory; the long term which contains information on previous concepts, while the short-term contains knowledge on the current one, as illustrated in Figure 3.13. Both learners, one modelled on short-term memory and the other on long term memory, are considered when predicting the final outcome based on previous performances (Losing et al., 2016).

Adaptive Random Forest (ARF), is an extension of Random Forest (RF) (Breiman, 2001) developed for non-stationary data streams (Gomes et al., 2017). Similar to the *Ensemble of Restricted Hoeffding Trees* (Bifet et al., 2012), the base-models in this ensemble are *Hoeffding Tree* instances (Domingos and Hulten, 2000). The main inspiration behind the

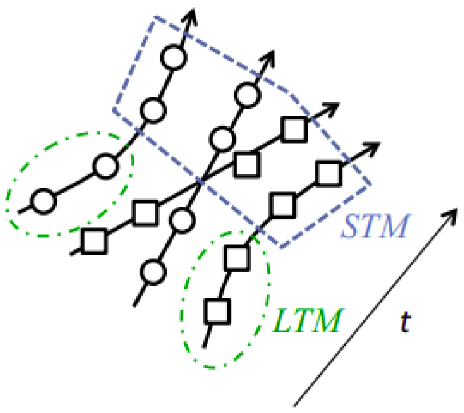


Figure 3.13: STM indicate short-term memory (current concept) and LTM indicate long-term memory (previous concepts) (Losing et al., 2016).

development of an adaptive variant of RF, was that this model requires very little intervention in terms of preparing input data and optimising the hyperparameters while also obtaining high performances (Gomes et al., 2017). Similar to the study conducted by Bifet et al. (2012), the features are randomly sampled during the training phase and ADaptive WINdowing (ADWIN) was proposed, to train background trees when a drift warning occurs. Once a drift is detected, background trees are replaced with the respective tree that triggered the warning, as shown in Algorithm 6 on lines 11 to 16. A key finding in this research was that there was no degradation of performance when comparing a parallelised version of ARF against a serial implementation, allowing for scalability in an evolving data-stream environment (Gomes et al., 2017).

Researchers also investigated the notion of adapting boosting ensembles to handle data-streams. One of the earliest algorithms employing boosting in this environment, was the *Pasting Votes* approach (Breiman, 1999). They proposed that for every boosting round, different subsets of data should be utilised and so, there is no need to persist all the data as in turn, this promotes adaptive learning. It was shown that this approach could handle datasets with up to a terabyte in size (Breiman, 1999). *Racing Committees* (Frank et al., 2002), a similar approach to *Pasting Votes*, includes a strategy called adaptive pruning in order to reduce time/memory cost, which is crucial in large datasets. A key point made in their study is that adaptive pruning not only influences time/memory cost but also, improves performance in terms of accuracy (Frank et al., 2002).

Inspired by *AdaBoost* (Freund and Schapire, 1997b), the *Learn++.NSE* ensemble han-

Algorithm 6 Pseudocode for ARF: **Symbols:** m: maximum features evaluated per split; n: total number of trees ($n = |T|$); δ_w: warning threshold; δ_d: drift threshold; $c(\cdot)$: change detection method; S: Data stream; B: Set of background trees; $W(t)$: Tree t weight; $P(\cdot)$: Learning performance estimation function (Gomes et al., 2017).

```
1:  function AdaptiveRandomForests($m, n, \delta_w, \delta_d$)
2:      $T \leftarrow CreateTrees(n)$
3:      $W \leftarrow InitWeights(n)$
4:      $B \leftarrow \varnothing$
5:      while $HasNext(S)$ do
6:          $(x, y) \leftarrow next(S)$
7:          for all $t \in T$ do
8:              $\hat{y} \leftarrow predict(t, x)$
9:              $W(t) \leftarrow P(W(t), \hat{y}, y)$
10:             $RFTreeTrain(m, t, x, y)$              ▷ Train $t$ on the current instance $(x, y)$
11:             if $C(\delta_w, t, x, y)$ then                         ▷ Warning detected ?
12:                 $b \leftarrow CreateTree()$                   ▷ Initialise background tree
13:                 $B(t) \leftarrow b$
14:             end if
15:             if $C(\delta_d, t, x, y)$ then                           ▷ Drift detected ?
16:                 $t \leftarrow B(t)$                   ▷ Replace $t$ by its background tree
17:             end if
18:         end for
19:         for all $b \in B$ do
20:             $RFTreeTrain(m, b, x, y)$
21:         end for
22:     end while
23: end function
24:
25: function RFTreeTrain($m, t, x, y$)
26:     $k \leftarrow Poisson(\lambda = 6)$                    ▷ Fixed param to Poisson Dist.
27:     if $k > 0$ then
28:         $l \leftarrow FindLeaf(t, x)$
29:         $UpdateLeafCounts(l, x, k)$
30:         if $InstancesSeen(l) \geq GP$ then  ▷ Grace period pre recalculating heuristics (split test)
31:             $AttemptSplit(l)$
32:             if $DidSplit(1)$ then
33:                 $CreateChildren(l, m)$
34:             end if
35:         end if
36:     end if
37: end function
```

dles non-stationary data streams by fitting weak-learners on batches of different distributions and fuses each hypotheses in the underlying ensemble using a majority vote (weighted) (Polikar et al., 2001). This ensemble proved to work with different patterns of drifts (described in Section 2.3.6), and by using weighted samples incoming from a stream, base-learners are penalised (misclassification) or rewarded (correct classification) by being attributed a higher weight based on their outcome. One drawback in this model is that complexity increases linearly over time, as the maximum number of base-learners is unbounded so as to handle reoccurring drifts (Polikar et al., 2001). Variations of this algorithm were proposed by Ditzler and Polikar (2013), in order to handle both data-imbalance and concept drift. *Learn++.CDS*, incorporates SMOTE (Chawla et al., 2002) with the previously mentioned algorithm to tackle imbalance, while *Learn++.NIE* utilises bagging based sub-ensembles to do so (without generating synthetic instances). Although this model handles both concept drift and class imbalance, it was shown to have a significant impact on complexity, when compared against other adaptive learners, including *Streaming Ensemble Algorithm* (Ditzler and Polikar, 2013).

A more recent adaptive algorithm was proposed by Montiel et al. (2020), were they formulated an adaptation of eXtreme Gradient Boosting (XGBoost) to work with non-stationary data streams. The key idea behind this method is the proposed approach employed to update or create base-learners (weak) in the underlying ensemble. Their approach works as follows (Montiel et al., 2020);

1. Gradient boosting or more specifically XGBoost with decision trees is used as an ensemble.

2. In a traditional batch environment, the ensemble E (gradient boosting), is built iteratively, wherein each iteration k, a base hypothesis function f_k is added to the ensemble with the objective of minimising the overall loss denoted as l, as shown below.

$$l(E) = \sum_{k=1}^{K} l(Y, \hat{Y}^{k-1} + f_k(X)) + \Omega(f_k)$$

3. K is the number of base-learners (typically regression trees) in E, whereas $f_k \in F$, where F denotes all possible space for the base hypothesis function. The predicted outcome $\hat{Y}$, is the summation of all the base hypothesis functions, which is considered as a step-wise constant. The Ω parameter is employed to penalise complex functions (regularisation).

4. In E trees are created in an additive manner, where at each k, a tree is appended to the ensemble, based on the score Y^k of previously added trees (descending gradient in a functional space).

5. The final predicted outcome/vote of E, is the sum of each base hypothesis function f_k, denoted as $\hat{y}_i = \sum_{k=1}^{K} f_t(x_i)$

6. Instead of using the whole training set to select each hypothesis function f_i, they proposed to append incoming instances into a buffer and once the buffer is full a single f_k is fitted. The succession of filling up the buffer and resetting it once a batch is formulated, can be considered as tumbling windows, denoted as $w = (\vec{x_i}, y_i) : i \in \{1, 2, ..., W\}$ where W is the maximum size for a specific window, $\vec{x_i}$ is a feature vector and y_i the respective label. The main idea is to pass w to the underlying base-learners f_{k-1} to direct the creation of f_k using their residuals. Therefore, the predicted outcome of the appended trees used to direct the creation of f_k, can be written as;

$$\hat{Y}^k = \hat{Y}^{k-1} + f_k(w_k)$$

In order to deal with concept drift, they proposed the following methods; since the ensemble is of a fixed size M, the underlying ensemble is updated when it is full, using; (i) *push strategy*: the oldest models are *removed* to make place for more recent ones (similar to a queue structure) (ii) *replace strategy*: oldest members are *replaced* by newer ones. (Montiel et al., 2020). These strategies are better illustrated in Figure 3.14. Dynamic windowing was also proposed due the performance of the ensemble being suboptimal at the beginning of the stream, since it needs to wait for the buffer to fill up, in order to create a complete ensemble. This concept works by setting an initial window size based on a minimum size, denoted as W_{min}, and double the current window size until it reaches an upper limit, denoted as W_{max}. This is therefore expressed as; $W_i = min(W_{min} \cdot 2^i, W_{max})$. The use of a drift detector, in particular ADWIN, was also investigated in order to combat fast drifts (updates are triggered by the detector rather than with each new batch) (Montiel et al., 2020).

These proposed approaches were then evaluated against each other, and other adaptive learners including but not limited to; *Accuracy Weighted Ensemble*, *Adaptive Random Forest*, *Leverage Bagging*, *Self Adjusting Memory*, *Online Bagging*, *Ensemble of Restricted Hoeffding Trees* and *Hoeffding Adaptive Tree*. From their empirical evaluation, it was noted that the Adaptive eXtreme Gradient Boosting (AXGB) without a drift detector (passive approach) in conjunction with *replacement strategy*, outperformed other versions of

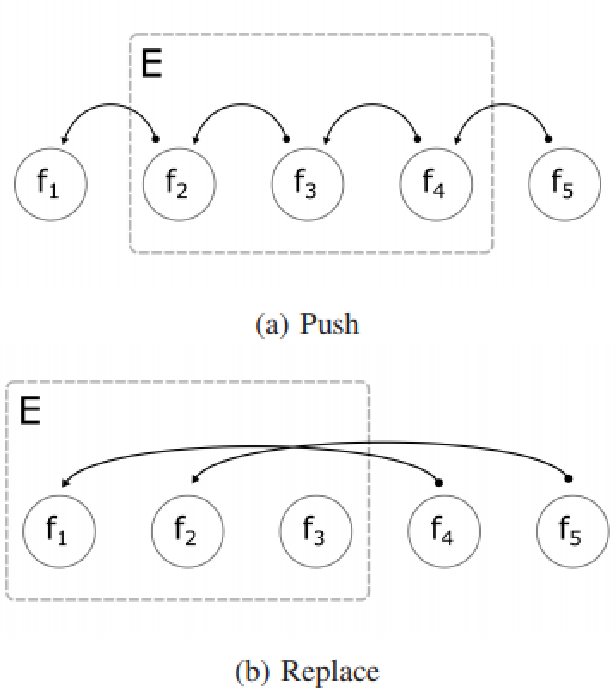


Figure 3.14: Two updating mechanism in AXGB used to update ensemble when it is full (Montiel et al., 2020).

AXGB. An overhead can be caused when using a drift detector, as it is typically traded-off for performance, however, in this scenario the detector-less model, performed better than the one with a detector. When compared to the other models, it performed relatively the same as *Adaptive Random Forest*, *Ensemble of Restricted Hoeffding Trees*, and *Leverage Bagging*, all of which placed in the top tier (Montiel et al., 2020).

The use of hyperparameter tuning (Grid Search) was also investigated to find optimal parameters using the first 30% of the stream. To their surprise, another proposed adaptation of XGBoost coined *BXGB* outperformed all other proposed variants when tuning was applied (Montiel et al., 2020). This variant is simply an *ensemble of ensembles* where each sub-ensemble (instances of XGBoost) are amended with each new incoming batch of data, and the final predicted outcome is fused using a majority vote. Although this model showed potential when tuning was applied, they noted that in practice, this model is not feasible when resource consumption is a primary concern. It was stated that their adaptation of XGBoost is a promising model to handle evolving data-streams given its adaptability and efficiency (Montiel et al., 2020).

3.2.3.2 | Instance Incremental versus Batch Incremental Learning

These aforementioned adaptive algorithms designed to handle evolving data-streams, can be grouped into two categories; *Instance Incremental* and *Batch Incremental* algorithms (Read et al., 2012). In batch incremental, incoming instances from a data stream are batched in order to train the underlying adaptive learner, and this schema includes algorithms such as; *Pasting Votes*, *Learn++.NSE*, *Learn++.CDS*, *Learn++.NIE*, *Racing Committees*, *Accuracy Weighted Ensembles* and *Adaptive eXtreme Gradient Boosting (AXGB)*. On the other hand, instance incremental, refers to when adaptive models learn from each instance as it appears in a stream and this schema includes models such as; *Online Bagging*, *Leveraging Bagging*, *Ensemble of Restricted Hoeffding Trees*, *Adaptive Random Forest* and *Self Adjusting Memory* (Montiel et al., 2020; Read et al., 2012).

In incremental batch learning, adaptive learners are not able to learn sample by sample, therefore, batches are instead used for training. In this schema it is required to free up batches from memory, as these can accumulate over time and result in the consumption of resources. The same issue can occur if base-learners are appended with new models without deleting/replacing older models (Read et al., 2012). Also, as batches are formulated with every w incoming samples, this schema does not allow learning from recent instances until a batch is formulated. Choosing the right w (batch size) is crucial in this setting, as there needs to be stability between model performance which is achieved from larger w and response/adaptation which is obtained by setting a smaller value for w (Read et al., 2012). The optimal value of w is also reliant on the domain and the type of stream at hand.

Instance Incremental Learning has its own set of drawbacks. Only after exposure to a relatively substantial amount of samples, can these models learn a concept accurately. Otherwise, they can have a hard time learning new concepts given the fact that learning occurs with each sample (Read et al., 2012). Even though both schemas have their pros and cons, this schema is usually chosen over batch-incremental due to it being less resourceful and the ability of "indefinite" learning, which stems from the nature of incremental learning itself (Read et al., 2012). Empirical evidence reported by Read et al. (2012), show that both schemas perform relatively the same. These findings were also backed up by a recent study conducted by Montiel et al. (2020).

3.3 | Overview of Recent Literature

In this chapter, an overview of various machine learning techniques employed to handle the detection of illicit activities in traditional finance and the blockchain was given. Recent literature suggests that use of decision-tree based ensembles can obtain remarkable performance in detecting illegal activities on both transactional and account level (Baek et al., 2019; Bartoletti et al., 2018; Farrugia et al., 2020; Harlev et al., 2018; Lee et al., 2020; Liang et al., 2019; Lin et al., 2019; Monamo et al., 2016; Sun Yin et al., 2019; Toyoda et al., 2017; Weber et al., 2019; Zola et al., 2019). More specifically, the use of RF, where, in some cases, it even obtained a higher score than deep learning algorithms such as GCN (Weber et al., 2019). Another type of decision-tree based ensemble widely used in recent literature is gradient boosting (with decision-tree as base learners). A recent study showed that XGBoost can effectively identify illicit activities on the Ethereum Network (Farrugia et al., 2020), while others indicated that LGBM can obtain similar results on the Bitcoin Network (Lin et al., 2019). The use of graph structures is widely supported in this domain, whether to employ graph learning algorithms or extract valuable information to be used as features to supervised classifiers (Dorofeev et al., 2018; Gaihre et al., 2019; Phetsouvanh et al., 2018; Toyoda et al., 2017; Weber et al., 2019; Zola et al., 2019; Zola et al., 2019).

Various studies also made sure to tune the models accordingly to improve performance further while giving details of the mechanism used (i.e. Random Search, Grid Search and Manual Search) together with the parameter space used while employing such techniques (Farrugia et al., 2020; Harlev et al., 2018; Sun Yin et al., 2019; Weber et al., 2019), while others did not provide any information or default settings were used (Baek et al., 2019; Bartoletti et al., 2018; Lee et al., 2020; Liang et al., 2019; Lin et al., 2019; Monamo et al., 2016; Toyoda et al., 2017; Zola et al., 2019). Some of the studies also investigated the idea of applying conventional data-sampling techniques to counteract this, in particular, RUS, ROS or SMOTE (Bartoletti et al., 2018; Harlev et al., 2018; Sun Yin et al., 2019). However, these techniques are known to have their disadvantages (for example, creating noisy synthetic data), as discussed in detail in Section 4.2.2.2. The idea of handling concept drift is often left unhandled in the context of the identification of nefarious activities on the blockchain, even though studies focusing on traditional finance state that it is prevalent in this domain. Weber et al. (2019) suggested to use a variant of GCN (EvolveGCN) to handle the degradation performance over time when identifying illicit bitcoin transactions, however, even though the model shows potential in recovering from a shift in distribution, a much more simplistic model RF obtained an overall better result in terms of F1-Score, Recall and Precision.

This chapter intended to give an overview on the state-of-the-art approaches employed in our target domain. The next chapter will describe the techniques employed in our proposed solution, while justifying each decision for the design process based on this reviewed literature.

4

Methodology

This chapter describes the methodological design employed in our study, and the justifications for the design decisions taken, in providing a solution to our problem. In Chapter 3, we summarised and gave a critical analysis of existing literature conducted in terms of detecting money laundering and other financial crime, in both cryptocurrencies and financial systems. This knowledge served as a foundation to plan our investigation and aid in delivering our objectives described in Section 1.3. To better understand our proposed approach, we start this chapter by giving a detailed definition of the problem tackled in our study, followed by a full description of our multi-phased methodology.

4.1 | Account and Transaction Level Detection

This study involved investigating whether we could identify (i) *if transactions broadcasted on the blockchain, can be deemed as illicit or licit* (ii) *if an account on the blockchain, can be considered as illicit or licit*.

A specific transaction is considered as illicit (or licit), if the address(es) listed as an input for this transaction, relates to an entity (holds the private keys) considered as illicit (or licit). Our motive was to identify money laundering movements by catching suspicious transaction flows between these entities. With the ever-growing transaction volume (Chainalysis, 2020), all the possible types of transaction flows (illicit-illicit, illicit-licit, licit-illicit and licit-licit) and the pseudonymous nature of cryptocurrency, the identification of money laundering can be a complicated task to accomplish, becoming more complex when solely performed through manual investigation. Monitoring transaction flows in order to capture suspicious movements can lead to the identification of money laundering. The process of doing so is conducted by firstly distinguishing if an individual transaction is illicit or not and then, given that all information on the blockchain network is public, one can examine the movement of funds. Employing machine learning models can drastically reduce the time it takes to identify illicit payment flows, and this presents the opportunity of having a real-time system, which screens and captures these flows at a transactional level, by assessing the risk associated with them.

One other way to identify money laundering is to monitor activity on an account-level, however this method may require the overhead of monitoring account by account, which in turn could be impractical to apply in real-time (Lim et al., 2014). An account can be deemed as illicit if it operates or has ties to nefarious activities such as High Yield Investment Programs, scams, Ponzi-Schemes or financing terrorism. Unlike detecting transactions which are processed once (monitored) as the system screens them, accounts may need to be monitored individually, however, this paradigm may be more accurate with respect to detection rate. There are also indications that an illicit account is created for the intention of conducting an illegal activity, and remain dormant shortly after its served its purpose (Farrugia et al., 2020).

Despite the appeal of employing an account or transaction level detection system, it is a non-trivial task since most financial data suffers from class imbalance (Bartoletti et al., 2018; Ferreira et al., 2018; Fiore et al., 2019; Lahmiri et al., 2020; Lei et al., 2020) and concept drift (Somasundaram and Reddy, 2019; Yang and Xu, 2019). As discussed in Section 3.2.2, class imbalance is an intrinsic problem in financial data. A skewed class distribution may damage the effectiveness of machine learning models, thus making the hypothesis function harder to distinguish between the underrepresented samples (illicit transactions) and the over-represented ones. In addition to this, the system's effectiveness could be reduced over time. Since criminals often change their behavioural patterns to avoid detection, the data is subject to change. As explained in Section 3.2.3, a change in the statistical dynamics of the data, may affect the target function modelled on historical information, by producing inaccurate approximations, thus increasing errors as more recent samples are evaluated. These two problems have further negative ramifications, as they can also affect the performance of the system by increasing the False-Positive or False-Negative rates. The industry-standard has a high False-Positive rate of up to 90% (Weber et al., 2019), so we conducted our research with the motivation of decreasing the False-Positive rate, while keeping the False-Negative rate relatively low. In turn, this would reduce the manual work required to verify transactions or accounts (False-Positives), without allowing criminals to avoid detection (False-Negatives).

As discussed in Section 3.3, decision-tree based ensembles showed to effectively detect illicit activities in traditional finance and blockchain networks. A recent study reported that RF in conjunction with graph information, outperformed deep learning algorithms such as GCN in classifying illicit Bitcoin transactions (Weber et al., 2019).

Other studies (Bartoletti et al., 2018; Monamo et al., 2016) also reported that this model was the best performing in fraud and money laundering detection in cryptocurrencies. Hence, RF was specifically selected as a benchmark for our first objective.

Decision tree based gradient boosting showed to be effective in detecting money laundering in traditional finance (Dorofeev et al., 2018; Jullum et al., 2020), as well as detecting illicit activities on the blockchain (Lin et al., 2019). A recent publication (Farrugia et al., 2020) showed that gradient boosting effectively identified illicit activities on the Ethereum Network. However, to the best of our knowledge, research on gradient boosting in conjunction with graph information, to detect illicit activities on the blockchain was scarce. In addition, we found no previous comparative analysis of various state-of-the-art decision-tree based gradient boosting models in the context of detecting illicit activities on both an account and transaction level. Therefore, we chose to compare state-of-the-art gradient boosting algorithms against each other, as well as, against our chosen RF benchmark. The selected models were also used in conjunction with data-sampling techniques, to try and improve the detection of licit-or-illicit activities by reducing the impact of class imbalance.

Furthermore, from the reviewed literature, in particular Section 3.2.3.1, a recent study showed that gradient boosting can be adapted to handle concept drift (Montiel et al., 2020). Given these findings we have proposed another adaptation of gradient boosting, Adaptive Stacked eXtreme Gradient Boosting (ASXGB), as a means of handling concept drift more effectively, with the intention of improving detection rate. This will be discussed in further detail in Section 4.3.4.

Based on the more refined problem definitions outlined above, in conjunction with the reviewed literature, we update the objectives with their corresponding Hypotheses/Experiments shown in Table 4.1.

4.2 | Datasets

A vital aspect of testing our hypotheses was the datasets utilised. The pseudonymous environment of cryptocurrencies (Buterin et al., 2014; Nakamoto, 2009; Schwartz et al., 2014), makes it hard to identify the ground truth (capturing illicit to licit accounts/-transactions). Furthermore, finding data in a similar domain such as standard financial transactions, also proved to be challenging given the nature of its sensitive information. Research was carried out in order to find potential datasets fitting our problem, and after exploring various options and avenues, we adopted these three datasets - *'Elliptic'*

Objective(s)	Experiment	Hypothesis
Objective. 1	Experiment 1 (Section 5.2.4.1)	*1. We test the hypothesis of whether decision-tree based gradient boosting can improve on the classification of licit-or-illicit activities on both account and transaction level detection, when compared to Random Forest (RF) (extension of bootstrap aggregation).*
Objective 2	Experiment 2 (Section 5.2.4.2)	*2. We test the hypothesis of whether decision-tree based gradient boosting, in conjunction with data-sampling techniques, can further improve on the classification of licit-or-illicit activities detection at a transactional-level.*
Objective 3	Experiment 3 (Section 5.2.4.3)	*3. We test the hypothesis of whether our proposed Adaptive Stacked eXtreme Gradient Boosting (ASXGB), developed to handle concept drift, can further improve on the classification of licit-or-illicit transactions, in a stream environment.*

Table 4.1: Objectives and their corresponding experiment and hypothesis.

(Elliptic, 2020; Weber et al., 2019), *'Ethereum Illicit Accounts'*[1] (Farrugia et al., 2020), and *'NOAA'*[2] (Elwell and Polikar, 2011) datasets.

The researchers (Elliptic, 2020; Weber et al., 2019) who provided the *'Elliptic dataset'* argue that it constitutes the largest labelled (illicit vs licit) transactional data in any cryptocurrency; thus, this was selected as our primary benchmark dataset for transaction level detection, employed in all three experiments, for the following reasons:

- It includes a substantial amount of labelled (illicit vs licit) transactions, needed to train/evaluate our models.

[1] Ethereum Fraud Dataset: `https://github.com/sfarrugia15/Ethereum_Fraud_Detection`
[2] NOAA Dataset: `http://users.rowan.edu/~polikar/nse.html`

- It includes the flow of payments as the data was modelled as a transactional graph, which is invaluable in money laundering.
- It includes temporal information, which allowed us to explore the dynamics over time.

The *'Ethereum Illicit Accounts dataset'* (Farrugia et al., 2020) which comprised of illicit ethereum accounts was employed as the primary dataset for account level detection. The reasons behind this are:

- Evaluating our models on detecting illicit activities on an account level.
- Evaluating additional models which were not validated in the original paper (Farrugia et al., 2020).
- Evaluating models on a different cryptocurrency network as the dataset was made up of ethereum accounts.

Lastly, due to a shortage of publicly available data, we opted to evaluate our models on another dataset the *'National Oceanic and Atmospheric Administration (NOAA) Weather dataset'* (Elwell and Polikar, 2011). This dataset is commonly used to test models in concept drift environments (Cristiani et al., 2020; Liao et al., 2016; Montiel et al., 2020). This helped us validate how well the applied models can generalise to handle concept drift, on other real-world datasets.

4.2.1 | Datasets Description

The *Elliptic dataset* was downloaded from Kaggle[3], and it was provided in three different files, all of which were in CSV file format. These files consisted of transaction classes, features and an edge list, and when combined, they formed a transaction graph obtained from the Bitcoin blockchain.

Each transaction was considered as a node[4], with directed edges representing the flow of payments. Every node had 166 attributes which fell under two categories; local and aggregated features. A total of 94 local features included information such as output volume, transaction fee, number of outputs/inputs and timestep. These local features also included statistical measures such as the mean of outgoing/incoming transactions linked with the outputs/inputs.

[3] Elliptic Dataset: `https://www.kaggle.com/ellipticco/elliptic-data-set`
[4] This refers to a node/vertex within a graph structure

For the aggregated features, a total of 72 attributes were utilised, which included aggregated information from one-hop backward/forward from the centre node, such as standard deviation, minimum, maximum and correlation coefficients of neighbouring transactions, for the same information extracted for the local features (for example, transaction fee and inputs/outputs). Although the researchers (Weber et al., 2019) provided a brief description of the features, the exact description for each attribute was not provided due to intellectual property concerns.

A timestep was associated with each node. This attribute represented an estimated time for when the transactions were confirmed on the network. In total, there were 49 timesteps equally distributed within approximately two weeks, which can be extrapolated to cover roughly 98 weeks. However, it is worth noting that the researchers did not mention the exact period (from and to date) covered in this dataset. Each step comprised of a single connected component of transactions, which settled on the network within three hours or less from each other. No connecting edges existed which connected nodes with different timesteps.

Overall there were 203,796 transaction nodes with a total of 234,355 edges connecting these nodes. From all these transactions, 46,564 were labelled, with the remaining being undefined. The labels represented whether the entity executing the transaction (owning the private keys linked to the input address for a transaction), belonged to an illicit or licit entity. Examples of illicit entities included in this dataset were; Ponzi schemes, terrorists, ransomware, scams and malware, and examples for licit entities included, were; miners, wallet providers and exchanges. In total there were 4,545 transactions marked as illicit and 42,019 marked as licit. In this study, any unlabelled data was disregarded as we opted to investigate supervised machine learning algorithms, similar to the original study. The exact method used for the labelling process was not mentioned in the paper; however, they stated that a heuristics-based reasoning approach was employed to label such transactions.

The other two datasets also came in CSV file format. The *Ethereum Illicit Accounts dataset* consisted of a total of 4,681 instances, with 2,179 deemed as illicit and 2,502 as licit accounts. The dataset had a total of 42 features which were based on the transaction history for specific Ethereum accounts. These included aggregated information capturing the lifetime activity for a specific account, such as the total number of sent/received transactions and the minimum/maximum value ever received/sent. It also included information related to duration periods such as the average time in minutes between sent/received transactions and the time difference in minutes between the first and last transaction. A full detailed description of these features is shown in Appendix A. The *NOAA dataset* was compiled based on weather measurement from over 7k weather stations. The dataset comprises of a total of 18,159 readings, with 12,461 indicating no rain and the other 5,698 indicating rain. A total of 8 features were present in each sample; these included: Temperature, Dew Point, Sea Level Pressure, Visibility, Avg. Wind Speed, Max Sustained Wind Speed, Max Temperature and Min. Temperature. This dataset was indexed by time, where each tumbling window of 30 readings represented approximately one month of readings. It is worth noting that this dataset is known to be affected by concept drift (Elwell and Polikar, 2011).

4.2.2 | Analysing the Benchmark Transaction-Level Dataset

At this stage, we analysed the benchmark transaction-level dataset and performed various experiments. These experiments allowed us to make strong assumptions about the stationarity of this time series dataset, to better understand how the state of the data changed through time.

4.2.2.1 | Non-Stationary Temporal Data

In the original paper (Weber et al., 2019) that publicly provided the selected benchmark dataset, the researchers have noted that the models performance suddenly degraded at one point, due to the closure of a popular dark marketplace. This unexpected event occurred during the test time span, more specifically at timestep 43 as shown in Figure 4.1 The degradation in performance was an indicator that the dataset was non-stationary due to changing events. Hence we decided to conduct further analysis to determine the data stationarity for the selected dataset. Multiple tests were applied to detect time-series stationarity including; time plot tests and a unit root test.

The time plot tests helped us to visually capture how the mean and standard deviation for each feature changed over time. This procedure was performed as follows; we partitioned the dataset by class, denoted as $X_{illicit}$ and X_{licit}, as each group can have its own dynamics. For each feature x_i, in each partition $x_{illicit_i}$ and x_{licit_i}, we computed the mean $\overline{x}$ and the standard deviation s at each timestep t, thus extracting individual time-series representing the $\overline{x}$ and s of the partitioned features. The time-series was then plotted to capture the change in $\overline{x}$ and s visually. When we examined the resulting plots, it was shown that the mean $\overline{x}$ and the standard deviation s was drastically changing (that is, sudden spikes and irregular fluctuations) at each timestep t. In Figure 4.2 we show these changes for four specific features of the the partition containing illicit instances. These four features represent the attributes that exhibited the highest variance across the standard deviation. We visually show that the standard deviation was changing over each timestep t. Similar results were obtained when we analysed

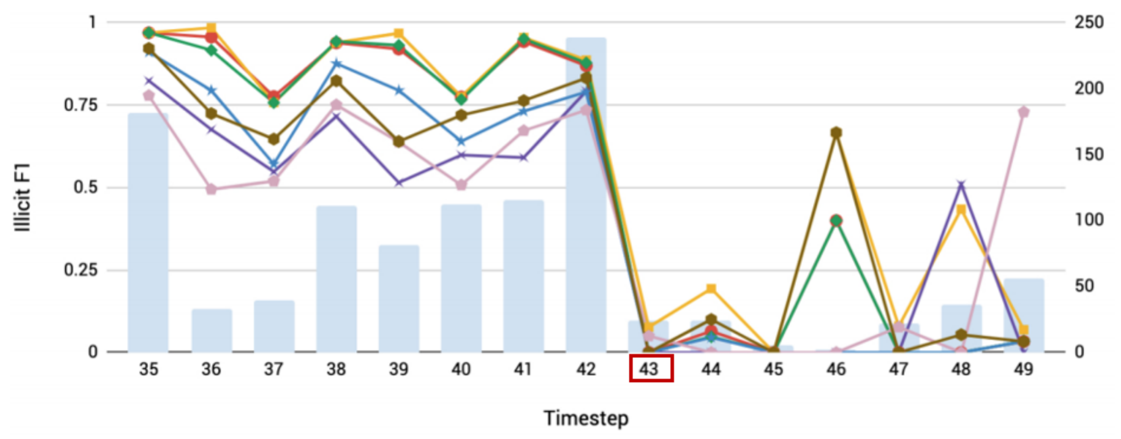


Figure 4.1: Models performances degraded due to the sudden closure of a dark market-place (timestep 43 marked in a red square) as reported by Weber et al. (2019).

the other partition for the licit instances. The resulting visual representation of each time-series, seemed to indicate that a stationary process did not generate these series. To summarise, these plots indicated notable trends, and fluctuating levels in s and $\overline{x}$, however, some plots indicate that these values revert around a constant.

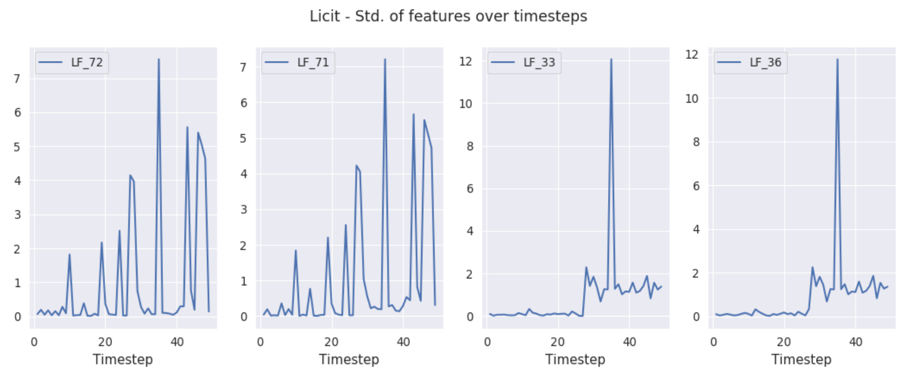


Figure 4.2: Time plots for the partition containing licit features, displaying the standard deviation across each timestep ($t_1 ... t_{49}$). In this plot, we only display the top four features with the highest variance across the standard deviation indexed by t.

These plots helped us to visualise changing levels and trends in our time-series. In addition, further testing was carried out by applying a statistical hypothesis test, which in turn aided us in making stronger assumptions about the stationarity of the data. For this experiment, the unit root test called the Augmented Dickey-Fuller test (ADF) (MacKinnon, 1994, 2010) served as a hypothesis test to check for non-stationarity. This involved testing whether an autoregressive model of the time-series has a unit root (stochastic trend), formulated as follows;

H_0 (Null Hypothesis): A unit root is present in the time-series (non-stationary)

H_1 (Alternate Hypothesis): A unit root is not present in the time-series (stationary)

The interpretation of the outcome uses the resulting p-value (95% confidence interval) to determine;

p-value > 0.05: Time-series is non-stationary as it has a unit root (Failed to reject H_0)

p-value <= 0.05: Time-series is stationary as it does not have a unit root (Reject H_0)

Similar to the previous experiment, we partitioned the dataset by class and then computed the mean for each feature at each timestep. Then we applied the ADF test for each resulting time-series. Table 4.2 and Table 4.3 show the highest p-values obtained for both licit and illicit partitions during this test. From a total of 165 individual time-series of features, we failed to reject H_0 for 61 features when tested on the licit partition, and 49 for the illicit partition. From the previous visualisations and the results gathered from this hypothesis test, it was concluded that the statistical properties of the benchmark dataset change over time.

Feature	p-value
AGG_26	0.985
AGG_42	0.973
AGG_9	0.959
AGG_6	0.947
AGG_7	0.939
AGG_61	0.854
LF_65	0.827
LF_59	0.827
LF_66	0.818
LF_60	0.818

Table 4.2: Top 10 highest p-values (failed to reject H_0) obtained when applying the ADF test on every feature, on the licit partition. The exact definition for the following features was not given due to intellectual property concerns (Elliptic, 2020; Weber et al., 2019).

4.2.2.2 | Class Distribution

At this stage, we analysed the class distribution of the benchmark dataset by plotting the ratio between illicit to licit (minority to majority) transactions across each timestep. The one month moving average for this ratio was also added to this plot, to smooth out weekly fluctuations and highlight a more distant change. Figure 4.3 displays the results captured during this test. From this visual representation, it was evident that

Feature	p-value
LF_1	1.000
AGG_38	1.000
AGG_33	1.000
AGG_31	1.000
AGG_14	1.000
AGG_37	0.999
LF_3	0.999
LF_5	0.999
AGG_2	0.996
AGG_55	0.995

Table 4.3: Top 10 highest p-values (failed to reject H_0) obtained when applying the ADF test on every feature, on the illicit partition. The exact definition for the following features was not given due to intellectual property concerns (Elliptic, 2020; Weber et al., 2019).

the skewed class distribution was not constant. These results, indicated that the rate of illicit transactions executed over the Bitcoin network varied over time.

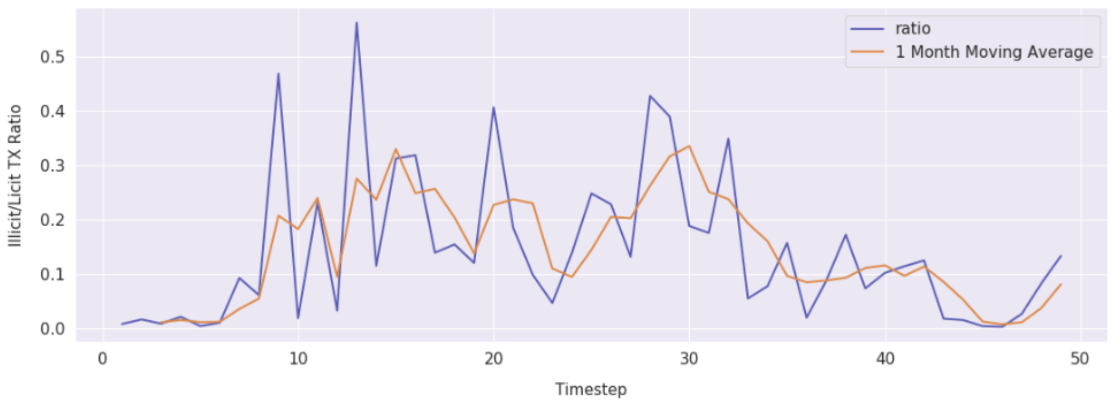


Figure 4.3: Illicit to licit ratio at each timestep in the benchmark dataset.

At this point, we analysed the dynamics of the benchmark dataset over time. Time plot tests visually showed that the mean and standard deviation for some features changed as time goes on. In the second hypothesis test, the ADF test (MacKinnon, 1994, 2010), statistically confirmed that there are non-stationary features present in the dataset. The final plot showed that the dynamics of the class distribution also changed over time, having an overall illicit to licit ratio of $\approx 1:9$. From these experiments, we concluded that concept drift was present in the selected benchmark dataset.

4.2.3 | Data Pre-Processing

The benchmark transaction-level dataset did not require any significant modifications, as it was reasonably fit for use to train machine learning models. As discussed in Section 4.2.1, the downloaded data came in three separate files; (i) node features (*'elliptic_txs_features.csv'*) (ii) node classes (*'elliptic_txs_classes.csv'*) (iii) edges between nodes (*'elliptic_txs_edgelist.csv'*).

Firstly, the node classes were loaded. These consisted of two columns; the transaction ID and its corresponding class. The encoding was altered as follows; '1' indicating illicit nodes, remained unchanged, '2' indicating licit nodes were changed to '0', and unlabelled nodes were disregarded. This resulted in a total of 46,564 labelled nodes. The next step involved loading up the features for each node. The CSV file did not include headers and so, the features needed to be manually labelled. Following the information found on Kaggle, the first column, 'transaction ID' was named - *'txId'*, and the following column 'timestep', was named - *'ts'*. The following 93 columns represented the local features (for example, output volume and transaction fee) which were explained in detail in Section 4.2.1, and these were renamed as *'LF_n'*, with *n* being the column index increased by 1. This resulted in columns *'LF_1'* to *'LF_93'*. The remaining 72 columns represented the aggregated features (aggregated information from one-hop backward/-forward from the centre node) which were also explained in detail in Section 4.2.1, and these were renamed as *'AF_n'*, with *n* being the column index increased by 1. This resulted in columns *'AF_1'* to *'AF_72'*.

In the benchmark paper (Weber et al., 2019), node embeddings were extracted from a Graph Convolutional Network (GCN) (Kipf and Welling, 2017), which were later exploited as feature sets. These embeddings were not available on Kaggle; therefore, we trained a GCN with the same documented hyperparameters (using altered public code found on GitHub[5]) to extract these embeddings ourselves.

This procedure consisted of; (i) loading up the dataset as a graph, which included combining features, classes and edges (ii) training a 2-Layer GCN using an Adam optimiser with a learning of 0.001 for 1000 epochs (iii) employing cross-entropy as a loss function with a weighted ratio of 0.3/0.7 (licit to illicit) (iv) extracting embeddings with a size of 100 from the last linear projection just before softmax logits[6]. These embeddings were then saved as *'elliptic_embs.csv'* and were made public on Google Drive[7]. The resulting file consisted of a transaction ID, followed by the embeddings named as

[5] GCN Implementation: `https://github.com/tkipf/pygcn`

[6] Embeddings Extraction: `https://github.com/tkipf/pygcn/issues/26#issuecomment-435801483`

[7] Node Embeddings: `https://drive.google.com/file/d/1RTuznxBk9_PdOETrKbsGAQH5jXj25qrB/view?usp=sharing`

'NE_n', where *n* was the column index increased by 1.

Ultimately, all the mentioned files were loaded up and merged with the transaction ID, resulting in one whole dataset. Keeping inline with the benchmark paper (Weber et al., 2019), this dataset could be loaded/evaluated on different feature sets, including; (i) using local features only, denoted as *'LF'* (ii) using all the features, which include *'LF'* and the aggregated features, denoted as *'AF'* (iii) using *'LF'* and node embeddings, denoted as *'LF'_NE'* (iv) using *'AF'* and node embeddings, denoted as *'AF_NE'*. These feature sets and their relevance to our study will be discussed in the upcoming Section 5.2.

The other datasets required minimal changes in order to be able to process and train our models. For the *Ethereum Illicit accounts dataset* (account-level benchmark dataset), only six additional features needed to be removed, as they were not present in the original study (Farrugia et al., 2020). Apart from that, empty values were substituted by 0, and categorical variables were numerically encoded. For the *NOAA* dataset, each instance was stamped with the corresponding timestep, and this was done by applying a tumbling window on every 30 samples as specified in the original publication (Elwell and Polikar, 2011).

4.3 | Proposed Solution

In this section, we give a detailed description of the models employed in our approach, along with an explanation of the methods used, to tackle the problems described in Section 4.1. A high-level depiction for the proposed system architecture, is shown in Figure 4.4. An overview of the code and structure for the final solution is also being made available.

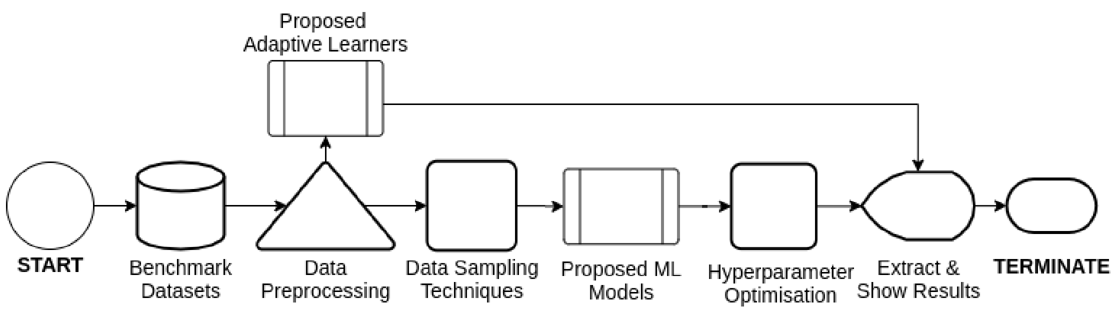


Figure 4.4: A high-level overview of our proposed solution in the form of a flow chart.

4.3.1 | Boosting Algorithms

Decision tree-based gradient boosting ensembles were chosen based on the following findings: (i) from the reviewed literature, the use of gradient boosting in conjunction with graph information to detect money laundering in cryptocurrency transactions, was limited (ii) previous studies investigating money laundering detection in traditional finance (Dorofeev et al., 2018; Jullum et al., 2020), showed the effectiveness of this ensemble, which in turn enticed further interest for investigation (iii) other types of decision tree ensembles, in particular Random Forest, proved to be the best performing model, in some cases also outperforming Deep Learning in money laundering and similar domains (Bartoletti et al., 2018; Monamo et al., 2016; Rokach, 2016; Weber et al., 2019) thus, introducing an opportunity to compare these decision-tree ensembles in the context of this problem.

In our study, we proposed to investigate the effectiveness of boosting algorithms in detecting money laundering in cryptocurrency transactions and so, three state of the art boosting algorithms were explored; eXtreme Gradient Boosting (XGBoost) (Chen and Guestrin), LGBM (Ke et al., 2017), and CatBoost (Prokhorenkova et al., 2018). All of these algorithms sequentially build decision trees as weak learners (high bias low variance) by fitting gradients, of residuals of previously created trees, and in turn, minimising the overall prediction error. The final predictions, are then made based on the consolidated estimates of the underlying weak learners. However, these different algorithms have their own set of characteristics outlined in Section 2.4.2. This further motivated us to investigate all three approaches and identify which gradient-boosting framework better suited our problem. We used the XGBoost, LGBM, and CatBoost algorithms found in the following libraries: xgboost[8] version 1.0.1, lightgbm[9] version 2.3.1, and catboost[10] version 0.20.2, respectively. Unless specified the default parameters for these algorithms are used.

4.3.2 | Handling a Skewed Class Distribution

The benchmark dataset suffers from class imbalance, as stated in Section 4.2.2.2; therefore, we applied various Data-Level techniques to counteract this. We selected these techniques instead of Algorithm-Level approaches due to the constraints described in Section 3.2.2.2.

[8] xgboost: `https://xgboost.readthedocs.io/en/latest/python/index.html`
[9] lightgbm: `https://lightgbm.readthedocs.io/en/latest/`
[10] catboost: `https://catboost.ai/docs/concepts/python-quickstart.html`

As stated in Section 3.2.2, when dealing with a skewed class distribution, there is no "one size fits all" approach. Keeping this in mind, as well as considering our time constraints, we investigated sampling techniques that seemed to be worth pursuing given the reviewed literature. The following sampling approaches were applied: Synthetic Minority Over-Sampling (SMOTE) (Chawla et al., 2002), Neighbourhood Cleaning Rule (NCL) (Laurikkala, 2001) and NCL-SMOTE (Junsomboon and Phienthrakul, 2017). These techniques were compared against each other and ultimately, the non sampled dataset, in order to determine any potential improvement.

SMOTE was used as an oversampling technique as it proved to be effective in different domains (Harlev et al., 2018; Sayed et al., 2019; Yong Sun and Feng Liu, 2016); pseudocode for which was shown in Algorithm 3, in Section 3.2.2. However, as explained in Section 3.2.2.1, this approach could further increase noise, if noisy outliers were present in the dataset. Therefore, as a means of neutralising this issue, we investigated the use of NCL-SMOTE (Junsomboon and Phienthrakul, 2017), where NCL was applied to remove noisy outliers before oversampling the dataset using SMOTE. In a study conducted by Junsomboon and Phienthrakul (2017), this approach showed to increase the overall recall, thus reducing false-negatives. Given that the results published in the benchmark paper (Weber et al., 2019) suffered from high false-negatives in comparison to the false-positives, this approach seemed appropriate to investigate. NCL (Laurikkala, 2001), which is an approach that under-samples the dataset by cleaning up noise from the majority samples, was tested on its own. The decision to do so, was inspired by a recent publication (Tyagi and Mittal, 2020) whereby, NCL showed to outperform various over and under-sampling techniques, particularly on financial data.

All of these approaches were implemented in Python using the imbalanced-learn [11] library version 0.6.2 (Lemaître et al., 2017). The default parameters for each technique were applied to sample the training sets. We also published the sampled datasets on Google Drive[12].

4.3.3 | Hyperparameter Optimisation

The proposed models all underwent hyperparameter optimisation. This decision was taken since the Gradient Boosting Machine (GBM) algorithms posses a significant number of hyperparameters and so, tweaking these parameters could improve results (Farrugia et al., 2020; Jullum et al., 2020; Prokhorenkova et al., 2018; Xia et al., 2017). As for the selected approach, we opted to make use of Bayesian hyperparameter optimisa-

[11] imbalanced-learn: `https://imbalanced-learn.org/stable/`
[12] Sampled Datasets: `https://drive.google.com/drive/folders/1xxJgmMPKVGLymI9OfX1JxHFU9GCEJvK-`

tion, more specifically, the Tree-structured Parzen Estimator (TPE) (Bergstra et al., 2011), since it showed its efficiency in high dimensional search spaces (Prokhorenkova et al., 2018; Xia et al., 2017), when compared to other approaches (Xia et al., 2017). This was implemented using Optuna[13] library version 1.3.0 (Akiba et al., 2019).

The study carried out by Prokhorenkova et al. (2018), heavily influenced our procedure to search for the optimal hyperparameters. Our decision was mainly due to their use of TPE and their comparison against the same selected GBM algorithms. In order to apply this technique the following steps were implemented;

1. The dataset was split into in-sample (70%) and out-of-sample (30%) sets
2. The sequential optimisation algorithm, TPE, was applied for 50 steps
3. In each step, denoted as i, the in-sample set was further split into k-folds ($k = 5$) using a stratified split by label
4. Cross-validation was applied by; setting the fixed hyperparameters outputted by the TPE at i, and applying an exhaustive search on the number of trees in the ensemble (1 to 5000), denoted as n, on each k
5. The F1-Score was collected for each n on each k. The scores were then averaged over k, and the n with the $\max_{F1-Score}$ was selected for i
6. Steps 3 to 5 were applied until $i = 50$. The hyperparameters with the highest F1-Score from all the steps, were considered to be the most optimal

Taking into consideration our aim to reduce both the false-positive and false-negative rates, the objective function described in Prokhorenkova et al. (2018) approach [14], needed to be altered. We swapped the objective function from $\min_{logloss}$ to $\max_{F1-score}$. The ranges and specific hyperparameters explored during this procedure were also selected based on previous literature (Prokhorenkova et al., 2018; Xia et al., 2017). Table 4.4 shows the list of the hyperparameters tried for each GBM model.

4.3.4 | Handling Non-Stationary Temporal Data

Through our visual and statistical analysis in Section 4.2.2.1, we assumed with a high degree of confidence, that the benchmark transaction-level dataset exhibited concept drift.

[13] Optuna: A hyperparameter optimization framework: `https://optuna.readthedocs.io/en/stable/`
[14] CatBoost Quality Benchmarks: `https://github.com/catboost/benchmarks/tree/master/quality_benchmarks`

Parameter	**Range/Distribution**	**Description**
XGBoost - includes exhaustive search for number of gradient boosting trees [1,5000]		
learning_rate	Log-uniform [e^{-7}, 1]	Boosting learning rate
max_depth	Discrete-uniform [2, 10]	Max tree depth for base classifiers
subsample	Uniform [0.5, 1]	Subsample ratio of training instance
colsample_bytree	Uniform [0.5, 1]	Subsample ratio of columns when constructing each tree
colsample_bylevel	Uniform [0.5, 1]	Subsample ratio of columns for each level
min_child_weight	Log-uniform [e^{-16}, e^{5}]	Min sum for instance weight (hessian) in a child
reg_alpha	Log-uniform [e^{-16}, e^{2}]	L1 regularization
reg_lambda	Log-uniform [e^{-16}, e^{2}]	L2 regularization
gamma	Log-uniform [e^{-16}, e^{2}]	Min loss reduction for further partition on a leaf node
LightGBM - includes exhaustive search for number of gradient boosting trees [1,5000]		
learning_rate	Log-uniform [e^{-7}, 1]	Boosting learning rate
num_leaves	Discrete-uniform [2, 1000]	Max no. of leaves in a tree
subsample	Uniform [0.5, 1]	Subsample ratio of training instance
colsample_bytree	Uniform [0.5, 1]	Subsample ratio of columns when constructing each tree
min_child_samples	Discrete-uniform [1, 400]	Min number of data in one leaf
min_child_weight	Log-uniform [e^{-16}, e^{5}]	Min sum for instance weight (hessian) in a child
reg_alpha	Log-uniform [e^{-16}, e^{2}]	L1 regularization
reg_lambda	Log-uniform [e^{-16}, e^{2}]	L2 regularization
CatBoost - includes exhaustive search for number of gradient boosting trees [1,5000]		
learning_rate	Log-uniform [e^{-7}, 1]	Boosting learning rate
depth	Discrete-uniform [4, 10]	Depth of the tree
random_strength	Discrete-uniform [1, 20]	Amount of randomness used for scoring splits
subsample	Uniform [0.5, 1]	Subsample ratio of training instance
l2_leaf_reg	Log-uniform [e^{-16}, e^{2}]	L2 regularization
leaf_estimation_iterations	Discrete-uniform [1, 10]	Regulates how many steps are done when calculating leaf values
rsm	Discrete-uniform [0, 1)	Percentage of features to use at each split selection
Random Forest		
n_estimators	Discrete-uniform [100, 1000]	No. of trees in forest
max_samples	Uniform [0.0, 1]	% samples to train each base
max_features	Choice [sqrt, log2]	No. of features for the best split

Table 4.4: Searching space used in hyperparameter optimisation.

Therefore at this stage, we introduced an adaptation of the eXtreme Gradient Boosting (XGBoost) algorithm (Chen and Guestrin) to handle evolving data-streams, coined as 'Adaptive Stacked eXtreme Gradient Boosting (ASXGB)'; pseudocode for which is shown in Algorithms 7, 8, 9, 10 and 11. A recent study published by Montiel et al. (2020) heavily influenced our algorithm, as they showed that decision-tree based gradient boosting could effectively be adapted to create concept-drift-aware models. In the following explanation, we describe our procedure in the context of binary classification, that is $y \in \{c_1, c_2\}$.

Our proposed model is an ensemble of ensembles, which incrementally learns from batched data (*batch-incremental*) through a continuous data-stream. The underlying ensemble employs stacking (Wolpert, 1992), which is a meta-learning technique that was previously discussed in Section 2.4.3. In the context of our algorithm, stacking is used to combine multiple XGBoost classifiers (base models) denoted as $h_1, h_2, ..., h_N$, where N is the specified number of base-models in the ensemble, into one. This is done by feeding the base models outputs as an input to another model, commonly referred to as the meta-model. In this algorithm the meta-model is another XGBoost classifier and we denote this as h_{meta}.

In our experiments we trained our ASXGB on a data-stream, that we denote as $A = (\vec{x}_t, y_t) : t \in \{1, 2, ..., T\}$, where A is the incoming stream, t is time, $T \rightarrow \infty$ (an unbounded stream), $\vec{x}_t$ is a feature vector at t, and y_t its respective label. Our proposed algorithm takes the continuous stream A, and further divides it into equally sized batches of contiguous non-overlapping time-windows. This is done by consuming incoming instances from the stream A and storing them into a buffer until the specified window size is reached, at which point a new training batch, w, is ready for our algorithm (shown in Algorithm 7, lines 8 to 12). Once our algorithm is injected with this new batch, w, this batch is further split into two datasets (in our experiments we made use of a 60/40 ratio), which can be denoted as w_{base} and $w_{remainder}$. A new base model is then trained on w_{base} and then added to the stacked ensemble. If the number of base models found in the underlying ensemble is greater than 1, any previously added base-models will continue training on w_{base}. The continuation of training is carried out by applying the following process for each base model; build a new model as a continuation of the previously generated model, by appending new trees.

A new dataset (meta-dataset), D_{meta}, is then generated based on the predictions (probability distribution over the set of classes) for each base-model, when evaluated against $w_{remainder}$. Therefore, the features of D_{meta} comprises of the outputs of the base-models with the respective true label found in $w_{remainder}$. The meta-model, h_{meta}, is then trained, or if the number of base models is greater than 1, it proceeds with the contin-

Algorithm 7 Pseudocode for the Proposed ASXGB's Partial Fit Function

```
1: global variables
2:     $win_{size}$, Window Size
3:     $X_{win\_buf} \leftarrow [\,][\,]$, Window Buffer for $X$ (training)
4:     $y_{win\_buf} \leftarrow [\,]$, Window Buffer for $y$ (training)
5: end global variables
6:
7: function PartialFit($X, y$)                ▷ Partially (incrementally) fit the model
8:     $X_{win\_buf} \leftarrow storeBuffer(X_{win\_buf}, X)$
9:     $y_{win\_buf} \leftarrow storeBuffer(y_{win\_buf}, y)$
10:    while $nrow(X_{win\_buf}) \geq win_{size}$ do
11:        $X_{batch} \leftarrow X_{win\_buf}[0 : win_{size}]$
12:        $y_{batch} \leftarrow y_{win\_buf}[0 : win_{size}]$
13:        $trainBatch(X_{batch}, y_{batch})$
14:        $X_{win\_buf} \leftarrow removeBuffer(X_{win\_buf}, 0, win_{size})$
15:        $y_{win\_buf} \leftarrow removeBuffer(y_{win\_buf}, 0, win_{size})$
16:    end while
17: end function
```

uation of training on D_{meta}, to form a final generalised classifier. Once h_{meta} is trained, each base model proceeds with the continuation of training on $w_{remainder}$, in order to fully expose the base-models to the current data batch, without violating the principle of training the meta-model on unseen data. Given that the data stream can potentially be ∞ and evolving, an update mechanism is put in place to constantly update the ensemble once it's full (the total number of base-models is equal to N). This procedure is as follows;

1. For every round denoted as k, that h_{meta} is trained on D_{meta}, take note of feature importance (each feature's contribution for each tree in the meta-model, known as normalised gain).

2. Since the features passed are composed of the probability for the set of classes by each base-model, the feature importance corresponds to how important a specific h is, in the generalised classifier. Every h generates two features in a binary classification setting, therefore, the corresponding importance is added to indicate the overall performance of h, denoted as h^{gain}, noted at each k.

3. Since h is added incrementally with every new w, the total number of rounds (continuation of training), where h^{gain} is extracted from h, denoted as h^{rounds}, differ.

4. When the ensemble is full, any h with $h^{rounds} \geq n^{rounds}$ is selected for potential replacement. For any h that meets this criteria, the summation of the previous h^{gain}

for n^{rounds} is computed and the one with the minimum importance is replaced by a new base-model.

The training and updating mechanism for the described technique can be shown in pseudocode format in Algorithms 8, 9 and 10. It is worth noting that the number of features passed on to h_{meta} is fixed at $N * 2$, as XGBoost requires the same number of features when the continuation of training occurs. Since a new model is added incrementally to the ensemble when a batch is formulated, some attributes are passed as 0's until the ensemble is full; hence the performance of the stacked ensemble can be sub-optimal during the first rounds of the data stream. To counteract this, any predictions made before the ensemble is full, is carried out based on a majority vote of the already present base-models, in the ensemble. Alternatively, when the ensemble is full, the predictions of h_{meta} are used instead. Both of which are shown in Algorithm 11.

Montiel et al. (2020) inspired the idea of buffering a data-stream and splitting it into tumbling windows to train new XGBoost learners in the underlying ensemble. However, the update mechanism used to swap newer models within the ensemble and the prediction function, differ from our solution since we opted to combine the ensemble via stacking. However, ASXGB deviates from the standard-setting employed in stacking, where the overall problem is tackled by using multiple heterogeneous models on a single dataset. The idea is that each model is capable of learning a subspace of the overall problem. An intermediate prediction is then built from such models to train a meta-model. The final generalised model is then able to learn the whole space of the problem (Džeroski and Ženko, 2004). However, in our solution, we make use of homogeneous learners, where each model learns from a data stream via tumbling windows (rather than a single dataset). The intuition is that each incoming batch of data is representative of the current state of the evolving problem. Therefore, each model which is added/updated in the ensemble, would be capable of learning a subspace of the current state of the problem. The meta-model is then able to assess which of the underlying base-models is deviating away from the current state (degrading feature-importance through time); hence they will be swapped with newer models. Moreover, since it was shown that instance-incremental learning can on average be more efficient in terms of performance than batch-incremental learning, both can obtain similar performance in some cases (Read et al., 2012). Given the nature of XGBoost, batch-incremental learning was adopted. Data-Sampling and Hyperparameter tuning methods were not applied on this particular model, due to time constraints. We only employed an adaptation of XGBoost out of all the decision-tree based gradient boosting algorithms described in Section 4.3.1, particularly for the same reason.

Algorithm 8 Pseudocode for the Proposed ASXGB's Train Batch Function

```
1:  global variables*                                   ▷ Includes all the global variables from Algorithm 7
2:      $meta_{ratio}$, Training Ratio for Meta Model                 ▷ $1 < meta_{ratio} < 0$
3:      $n_{base}$, No. of Level-0 Models                                      ▷ $1 < n_{base}$
4:      $xgb_{base\_models} \leftarrow []$, Array of Level-0 Instances
5:      $xgb_{meta}$, Meta Model Instance
6:      $xgb_{base\_models\_performance} \leftarrow [][]$, Previous performances for base-models
7:      $xgb_{base\_models\_rounds} \leftarrow []$, Current rounds for base-models
8:  end global variables
9:
10: function TRAINBATCH($X_{batch}, y_{batch}$)
11:     $J \leftarrow int((1 - meta_{ratio}) * nrow(X_{batch}))$
12:     $w_{base\_X} \leftarrow X_{batch}[0, J]$
13:     $w_{base\_y} \leftarrow y_{batch}[0, J]$
14:     $w_{remainder\_X} \leftarrow X_{batch}[J, nrow(X_{batch})]$
15:     $w_{remainder\_y} \leftarrow y_{batch}[J, nrow(X_{batch})]$
16:     $meta\_features \leftarrow updateEnsemble(w_{base\_X}, w_{base\_y}, w_{remainder\_X}, w_{remainder\_y})$
17:     $new_{base} \leftarrow newBaseModel()$                       ▷ Returns a new XGB instance
18:     $new_{base} \leftarrow new_{base}.fit(w_{base\_X}, w_{base\_y})$
19:     $meta\_features_{new} \leftarrow new_{base}.predictProba(w_{remainder\_X})$
20:     $meta\_features \leftarrow append(meta\_features, meta\_features_{new})$
21:     $booster \leftarrow new_{base}.getBooster()$
22:     $new_{base} \leftarrow new_{base}.fit(w_{remainder\_X}, w_{remainder\_y}, booster)$      ▷ Continuation of training
23:     $i \leftarrow length(xgb_{base\_models})$
24:     if $i = n_{base}$ then
25:         $i \leftarrow getWeakestBaseModelIndex()$
26:     else
27:         $meta\_features \leftarrow fillMissingMetaFeatures(meta\_features)$              ▷ With 0's
28:     end if
29:     $xgb_{base\_models_i} \leftarrow new_{base}$                      ▷ Add new base-model to ensemble
30:     $xgb_{base\_models\_performance_i} \leftarrow []$
31:     $xgb_{base\_models\_rounds_i} \leftarrow 0$
32:     if $length(xgb_{base\_models}) = 1$ then                  ▷ First time training meta-model
33:         $xgb_{meta} \leftarrow xgb_{meta}.fit(meta\_features, w_{remainder\_y})$
34:     else                                           ▷ Continuation of training for meta-model
35:         $booster_{meta} \leftarrow xgb_{meta}.getBooster()$
36:         $xgb_{meta} \leftarrow xgb_{meta}.fit(meta\_features, w_{remainder\_y}, booster_{meta})$
37:     end if
38: end function
```

Algorithm 9 Pseudocode for the Proposed ASXGB's Update Ensemble Function

```
1:  global variables*                    ▷ Includes all the global variables from Algorithm 7, 8
2:      $n_{rounds}$, Rounds to evaluate Level-0 Models              ▷ $1 < n_{rounds} \leq n_{base}$
3:  end global variables
4:
5:  function UPDATEENSEMBLE($w_{base\_X}, w_{base\_y}, w_{remainder\_X}, w_{remainder\_y}$)
6:      $meta\_features \leftarrow []$
7:      $feature\_importance \leftarrow xgb_{meta}.getFeatureImportance()$
8:      for $i \leftarrow 1$ to $length(xgb_{base\_models})$ do
9:          $prev\_performance \leftarrow feature\_importance[i * 2] + feature\_importance[(i * 2) + 1]$
10:         $j \leftarrow xgb_{base\_models\_rounds_i} \% n\_rounds$
11:         $xgb_{base\_models\_performance_{i,j}} \leftarrow prev\_performance$
12:         $xgb_{base\_models\_rounds_i} \leftarrow xgb_{base\_models\_rounds_i} + 1$
13:         $booster_i \leftarrow xgb_{base_i}.getBooster()$
14:         $xgb_{base_i} \leftarrow xgb_{base_i}.fit(w_{base\_X}, w_{base\_y}, booster_i)$
15:         $meta\_features_{base_i} \leftarrow xgb_{base_i}.predictProba(w_{remainder\_X})$
16:         $meta\_features \leftarrow append(meta\_features, meta\_features_{base_i})$
17:         $booster_i \leftarrow xgb_{base_i}.getBooster()$
18:         $xgb_{base_i} \leftarrow xgb_{base_i}.fit(w_{remainder\_X}, w_{remainder\_y}, booster_i)$
19:     end for
20:     return $\leftarrow meta\_features$
21: end function
```

Algorithm 10 Pseudocode for the Proposed ASXGB's Get Weakest Base-Model Function

```
1:  global variables*                    ▷ Includes all the global variables from Algorithm 7, 8, 9
2:  end global variables
3:
4:  function GETWEAKESTBASEMODELINDEX
5:      $worst\_idx \leftarrow 0$
6:      $worst\_performance \leftarrow 1$
7:      for $i \leftarrow 1$ to $length(xgb_{base\_models})$ do
8:          if $xgb_{base\_models\_rounds_i} < n\_rounds$ then
9:              $continue$
10:         end if
11:         $performance \leftarrow xgb_{base\_models\_performance_i}.sum()$
12:         if $performance < worst\_performance$ then
13:             $worst\_performance \leftarrow performance$
14:             $worst\_idx \leftarrow i$
15:         end if
16:     end for
17:     return $\leftarrow worst\_idx$
18: end function
```

Algorithm 11 Pseudocode for the Proposed ASXGB's Predict Function

```
global variables*        ▷ Includes all the global variables from Algorithm 7, 8, 9, 10
end global variables

function PREDICT(X)
    if $length(xgb_{base\_models}) < n_{base}$ then        ▷ Ensemble is not full, take majority vote
        $predictions \leftarrow takeMajorityVote(X, xgb_{base\_models})$
        return $\leftarrow predictions$
    end if
    $meta\_features \leftarrow []$
    for $i \leftarrow 1$ to $length(xgb_{base\_models})$ do
        $meta\_features_{base_i} \leftarrow xgb_{base_i}.predictProba(X)$
        $meta\_features \leftarrow append(meta\_features, meta\_features_{base_i})$
    end for
    $predictions \leftarrow xgb_{meta}.predict(meta\_features)$
    return $\leftarrow predictions$
end function
```

4.3.5 | Implementation Details

The code employed in the development of the proposed solution was written in Python version 3.7.6, with the scripts also being made public on GitHub[15]. We opted to structure our code as a package named *'cryptoaml'*, with multiple modules and objects to break down each process into manageable code. These modules are as follow;

- **'datareader'**: This module handled the reading, pre-processing, and splitting of data into train and test sets. For each data source described in Section 4.2.1, an object derived from the base implementation *'_BaseDatareader.py'*, was written. Each object had its unique functionality to execute the aforementioned tasks. The only functionality exposed by this module was a function called *'get_data()'*. This took the following arguments; data source name, a path to a .yaml configuration file (containing file paths for the dataset), and a dictionary of keyworded arguments consumed by the concrete implementation (**kwargs). An instance of the specified data reader based on the data source name passed, was then returned. Pandas[16] version 1.0.1 (pandas development team, 2020; Wes McKinney, 2010) and Scikit-Learn[17] version 0.22.1 (Buitinck et al., 2013; Pedregosa et al., 2011) were the two leading third-party libraries imported in this module, as the datasets were converted in the form of DataFrames, and split using sklearn.model_selection *'train_test_split()'* function.

[15] CryptoAml Repository: `https://github.com/achmand/aml-crypto-graph`
[16] pandas: `https://pandas.pydata.org/`
[17] sklearn: `https://scikit-learn.org/stable/`

- **'data_sampler.py'**: This was not a module in itself, but a python script which handled any data sampling techniques that were applied, to counteract class imbalance. The techniques employed and described in Section 4.3.2, were implemented using this script.

- **'tune'**: Any procedures related to tuning the hyperparameters, explained in detail in Section 4.3.3, were carried out by this module. The primary function exposed was *'tune_model()'*, which took a classifier, a training set and the properties (i.e. parameter search space) needed to tune the model, as arguments. Once this function found the optimal hyperparameters, a trained classifier with these parameters was returned.

- **'metrics'**: Holds various scripts which exposed functionality to evaluate the classifiers, such as; computing F1-Score, Precision, Recall and Confusion Matrix. It was also utilised to output results in the form of tables and plots, which are shown in upcoming sections. The main third-party libraries imported in this module were; Scikit-Learn, to compute evaluation metrics, Seaborn[18] version 0.10.0, and matplotlib[19] (Hunter, 2007) version 3.1.3 to plot graphs.

Finally, two main scripts, *'model_tuner.py'* and *'result_extractor.py'* imported this package. The *'model_tuner.py'* script was used to tune the specified classifiers and then persist the trained models on disk. On the other hand, the *'result_extractor.py'* was executed to load the persisted classifiers, evaluate and extract the results, and to persist the outcomes. Both scripts, read from a .yaml configuration file, in order to run the scripts with the specified properties (i.e. what evaluation metrics should be extracted). Jupyter Notebooks[20] were used to display all the results extracted at each experiment.

[18] seaborn: `https://seaborn.pydata.org`
[19] matplotlib: `https://matplotlib.org/`
[20] Jupyter: `https://jupyter.org/`

5

Evaluation and Results

5.1 | Benchmark Models

In order to compare and contrast our proposed solution's effectiveness when undertaking the problems outlined in Section 4.2.2, multiple benchmark models needed to be identified, to evaluate alongside our proposed models. The process leading up to the selection of the benchmarks described below, was as follows; (i) reviewed recent literature in the same or similar domain (ii) apply critical analysis on this literature (iii) select models which were deemed as best performing to be used as benchmarks.

5.1.1 | Random Forest

When evaluating our proposed boosting algorithms in a traditional batch learning environment (not involving data-streams), the results obtained were assessed against a Random Forest (RF) classifier (Breiman, 2001). Similar to the models described in Section 4.3.1, this method is also a decision tree ensemble; however, it combines base-learners using bootstrap-aggregation and can grow in parallel. This model builds multiple decision trees on random variants of the same data, by training individual trees on random subsets of features. The aim is to create uncorrelated grown decision trees (high variance with low bias) to minimise the variance, which in turn reduces error. An aggregation of the predictions made by the underlying individual sensitive/unstable trees (unpruned/grown deep), will be averaged out through voting. The quality of the split for individual trees is measured using Gini impurity in the implementation of this benchmark model. The pseudocode for RF was shown in Section 2.4.1, in Algorithm 1.

The intentions for selecting this method as the leading benchmark model were; (i) its computational efficiency due to parallelisation (ii) when compared to other approaches in several domains (Rokach, 2016), it proved to be the best performing, particularly in fraud and money laundering detection in cryptocurrencies (Bartoletti et al., 2018; Monamo et al., 2016; Weber et al., 2019) (iii) it outperforms more complex Deep Learning algorithms, such as Graph Convolutional Network (GCN) (Kipf and Welling, 2017), in detecting illicit activities over the Bitcoin network (Weber et al., 2019). This classifier was implemented using a third-party library, Scikit-Learn version 0.22.1 (Buitinck et al., 2013; Pedregosa et al., 2011).

5.1.2 | Adaptive Random Forest

The Adaptive Random Forest (ARF) model (Gomes et al., 2017) proved to be effective in evolving data stream environments (Boiko Ferreira et al., 2019; Montiel et al., 2020), therefore, considering we had to evaluate a modification of gradient boosting adapted for this environment (ASXGB), we assessed it against this model.

This incremental-learner is an adaptation of Random Forest (RF) (Breiman, 2001) modified to work with evolving data streams. The most important aspects of this learner are; (i) utilises resampling to produce diversity (ii) selects subsets of the features at random for node splits, thus inducing diversity (iii) each base-learner is equipped with a drift detector, in order to track drifting concepts on each member of the ensemble (selective resets in the event of drifts) (iv) the final prediction is based on a weighted majority vote. Hoeffding Trees (Domingos and Hulten, 2000) are used as base-learners, together with ADaptive WINdowing (ADWIN) (Bifet and Gavalda, 2007) as drift detectors. If a drift warning is detected, this technique allows for training trees in the background, to replace the active tree if a warning escalates to a drift. Pseudocode for the ARF was shown in Section 3.2.3, Algorithm 6. The implementation for ARF classifier was produced using Sklearn-multiflow[1] (Montiel et al., 2018) version 0.4.1, and the default parameters for this algorithm were used.

5.1.3 | Adaptive eXtreme Gradient Boosting

The Adaptive eXtreme Gradient Boosting (AXGB) algorithm (Montiel et al., 2020) is an adaptation of XGBoost developed to handle evolving data streams, which was the basis for our proposed solution described in Section 4.3.4; hence, it was fundamental to compare this algorithm to the proposed Adaptive Stacked eXtreme Gradient Boosting (ASXGB).

[1] Sklearn-multiflow `https://scikit-multiflow.github.io/`

This algorithm is very similar to ASXGB, as both are batch-incremental learners, consume the data-stream with the use of buffers and tumbling windows, and make use of an ensemble of ensembles with XGBoost being the base learner. However, the final predictions made by this algorithm is based on a majority vote taken by the underlying ensemble. Moreover, this algorithm takes a different approach when updating the ensemble (shown visually in Section 3.2.3, Figure 3.14), once it is full, the following occurs; (a) *push strategy*: older models are removed before appending newer models, similar to First In First Out (b) *replacement strategy:* older models are replaced with newer ones.

Both modifications of this algorithm (*push strategy* and *replace strategy*), together with Adaptive Random Forest (ARF) were used as benchmark models, when assessing ASXGB in an evolving data stream environment (handling non-stationarity). The code required for these implementations, as well as the hyperparameters needed for each modification, were taken from GitHub[2], which was provided by the original authors of the paper (Montiel et al., 2020).

5.2 | Evaluation

It is important to set out a plan to evaluate the previously mentioned models empirically. In this section, we describe; the setup deployed in our study, the performance metrics, the evaluation frameworks, the design of experiments and any statistical tests employed.

5.2.1 | Setup

We carried out all development for this research on Microsoft Azure Cloud Computing [3]. More specifically, on an H16m (high performance compute) instance with the following specifications; 16 core Intel Xeon E5-2667 v3 Haswell 3.2 GHz (3.6 GHz with turbo) with 224GB DDR4 RAM using Python version 3.7.6 and Linux Ubuntu 18.04 LTS as an operating system. The development of our proposed solution also required the use of various Python libraries.

5.2.2 | Performance Metrics

In order to determine the relative effectiveness of the tested models, we adopted the use of several performance metrics. In this section, we give the rationale for selecting

[2] AXGB Implementation: `https://github.com/jacobmontiel/AdaptiveXGBoostClassifier`
[3] Microsoft Azure: `https://azure.microsoft.com`

these metrics, together with their corresponding definition. These metrics were specifically selected based on the reviewed literature (Ali et al., 2015; Bartoletti et al., 2018; Ferreira et al., 2018; Junsomboon and Phienthrakul, 2017; Tyagi and Mittal, 2020; Weber et al., 2019), as they are measures which better capture the effectiveness of the evaluated models on datasets with a skewed class distribution.

5.2.2.1 | Confusion Matrix

The Confusion Matrix is a table (shown below) with four different measures (binary classification) indicating all the combinations for the actual and predicted values. The information extracted from this table includes; (i) True Positives (TP) indicating predicted illicit transactions which are actually illicit (ii) True Negatives (TN) indicating the predicted licit transactions which are actually licit (iii) False Positives (FP) indicating predicted illicit transactions which were not actually illicit (iv) False Negatives (FN) indicating predicted illicit transactions which were not actually illicit.

		Predicted	
		licit (0)	illicit (1)
Actual	licit (0)	True Negative	False Positive
	illicit (1)	False Negative	True Positive

5.2.2.2 | Precision

A derivative measure of the confusion matrix which can be computed as follows; $Precision = \frac{TP}{TP+FP}$, where TP is the True Positive Rate and FP is the False Positive Rate. In the context of our problem, this measure captures; the ratio between the transactions predicted as illicit, to the actual number of illicit transactions.

5.2.2.3 | Recall

This metric is also another derivative measure of the confusion matrix. Recall quantifies the number of correct illicit transaction predictions out of all the illicit predictions made. Unlike Precision, this measure indicates the missed illicit predictions. Such metric can be defined as follows; $Recall = \frac{TP}{TP+FN}$, where TP is the True Positive Rate and FN is the False Negative Rate.

5.2.2.4 | F1-Score

The F1-Score is the harmonic mean of Precision and Recall, denoted as; $F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$. This measure takes both the False Negatives and False Positives into account, however the F1-Score is unable to differentiate between good Precision or Recall. This is due to the symmetrical nature of the formula.

5.2.2.5 | F1-Score over Time

This metric is the F1-Score over time (indexed by timestep) and has been only employed on datasets with temporal information (*Elliptic and NOAA datasets*), in order to capture any deterioration over time (Weber et al., 2019). The outcome from this measure have been outputted in the form of a plot.

5.2.2.6 | Accuracy

Accuracy, defined as; $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$, where TP is the True Positive Rate, TN is the True Negative Rate, FP is the False Positive Rate and FN is the False Negative Rate. In our context, this measure captures the ratio of correct illicit or licit predictions out of all the evaluated transactions. We did acknowledge the reviewed literature outlined in Section 3.2.2, stating that this score is not appropriate in datasets with class imbalance, however, to keep inline with the primary benchmark paper (Weber et al., 2019) this measure was reported.

5.2.3 | Evaluation Framework

Different frameworks have been employed to evaluate both the proposed and benchmark models, namely, the Standard Evaluation (Train/Test Split) and the Prequential Evaluation. The latter approach evaluates adaptive learners on a data-stream which could potentially be infinite. Conversely, the Standard approach evaluates static models on a finite dataset which is split into train and test sets. Table 5.1 shows the datasets employed in the evaluation phase, together with the corresponding properties, experiment(s) and framework(s) applied.

	Datasets		
Property	**Elliptic**	**Eth. Illicit Acc.**	**NOAA**
N Instances	$46,564$	$4,681$	$18,159$
N Features	94 - LF 194 - LF_NE 166 - AF 266 - AF_NE	42	8
Class Imbalance	$\approx 1:9$	Balanced	$\approx 1:2$
Temporal	49 timesteps	No timesteps	606 timesteps
Experiment 1 (Standard Eval.)	✓	✓	✓
Experiment 2 (Standard Eval.)	✓	✗	✓
Experiment 3 (Prequential Eval.)	✓	✗	✓

Table 5.1: Dataset(s) details and their corresponding evaluation framework. The *'N Instances'* shows the number of instances and *'N Features'* correspond to the number of features. For the Elliptic dataset, four feature sets were used, 'LF', 'LF_NE', 'AF' and 'AF_NE' (explained in detail in Sections 4.2.1 and 4.2.3). The *'Class Imbalance'* indicates the ratio between the minority to majority classes, whilst the *'Temporal'* shows the number of timesteps found in each dataset. The last three rows indicate whether a dataset was used in the corresponding experiment, where '✓' indicates yes and '✗' indicates no.

5.2.3.1 | Standard Approach (Train/Test Split)

In this approach, we split the dataset into a 70/30 ratio, resulting in two different sets - training and test sets. This approach is known to be the "Standard" when evaluating machine learning models in a traditional environment. It has been employed in numerous studies (Ferreira et al., 2018; Jullum et al., 2020; Weber et al., 2019; Zhang and Trubey, 2019) discussed in Chapter 3. This framework has been explicitly selected to test our models in a static environment, keeping inline with the evaluation framework employed in the benchmark paper (Weber et al., 2019).

In our evaluation, any hyperparameter optimisation or data-sampling techniques applied were conducted on the test set. The data with temporal information was split based on timestep, so as not to break the sequence of time (Jullum et al., 2020; Weber et al., 2019). In the *Elliptic dataset*, instances with a timestep of $<=$ 34 were used for training, while the following 15 timesteps were used for testing. The *NOAA dataset* was split in a similar fashion, where instances with a timestep $<=$ 424 were used as training, and the following 182 timesteps used for evaluation. Conversely, the *Ethereum Illicit Accounts dataset* was split on the number of instances, rather than, the number of timesteps, since it did not have any temporal information. This dataset was split as follows; (i) stratifying the dataset by class (ii) taking the first 3276 instances for training (iii) taking the remaining 1405 instances for testing. Moreover, due to the nature of RF and the proposed gradient boosting classifiers being non-deterministic, anytime these models were evaluated, the evaluation underwent 100 iterations. The resulting outcome would then be averaged as a means of getting closer to the actual effectiveness of this model. The distribution(s) of results have been plotted via box-plots.

5.2.3.2 | Prequential Evaluation

This evaluation paradigm is usually employed when testing adaptive learners (Montiel et al., 2020). The Prequential evaluation (predictive sequential) (Dawid, 1984) allows us to evaluate our models in a streaming environment, and it works using the following steps; (i) split the available dataset into batches (in our case based on timesteps) (ii) train model on timestep t (iii) evaluate model on timestep $t+1$ (iv) apply the steps two and three to all subsequent timesteps (Dawid, 1984; Hidalgo et al., 2019). This framework enabled us to simulate an "infinite" datastream from a finite dataset, utilise the full dataset for evaluation (no holdout needed) and allowed for smooth plotting of a performance metric over time (Hidalgo et al., 2019). In order to compare with previous experiments using the standard approach, whenever we employ this framework, the overall measure after t corresponding to the timestep which splits the test set (standard approach) is also outputted. Ultimately, this evaluation will serve as a simulation to real-time transaction monitoring.

5.2.4 | Design of Experiments

In this section, the design of the experiments is presented, together with flowcharts visually showing the processes required for each experiment (Figures 5.1, 5.2 and 5.3). In each experiment, we applied non parametric statistical tests described in the studies of Demšar (2006a) and Demšar (2006b), to confirm if any significant statistical difference was obtained between the proposed and benchmark models over multiple datasets. In particular, the Friedman Chi-Squared test was employed to determine if any statistical difference was present between the evaluated models ($\alpha \leq 0.05$). This test was followed by the post-hoc Nemenyi test, implemented to pinpoint the models which were statistically different. It is worth noting that Experiment 1 and 2 were conducted in a traditional batch learning environment (Section 5.2.3.1), while Experiment 3 was conducted in a data-stream environment (Section 5.2.3.2). The following third-party libraries were imported to execute such tests; SciPy (Virtanen et al., 2020) version 1.4.1 and Orange[4] (Demšar et al., 2013) version 3.23.1.

5.2.4.1 | Experiment 1

In this experiment, we test hypothesis 1 as defined in Table 4.1. We hypothesise that decision-tree based gradient boosting can improve on the classification of licit-or-illicit activities on both account and transaction level detection, when compared to Random Forest (RF) (extension of bootstrap aggregation). This experiment showed whether gradient boosting outperformed RF when evaluated against several datasets. More importantly, we were able to verify whether our proposed models obtained better results when compared to the best-performing model (RF), provided by the principal benchmark paper (Weber et al., 2019). In order to do so, we replicated the implementation of RF on the *Elliptic dataset* using the suggested hyperparameters (50 estimators and 50 max features), to obtain similar results reported in the paper, as shown in Chapter 3, Table 3.2. Gradient Boosting algorithms (XGBoost, LGBM and CatBoost) underwent hyperparameter optimisation before being evaluated. As for the datasets, all three datasets mentioned above were employed in this experiment. In order to keep inline with the primary benchmark paper, all the feature sets mentioned in Section 4.2.3 were tested.

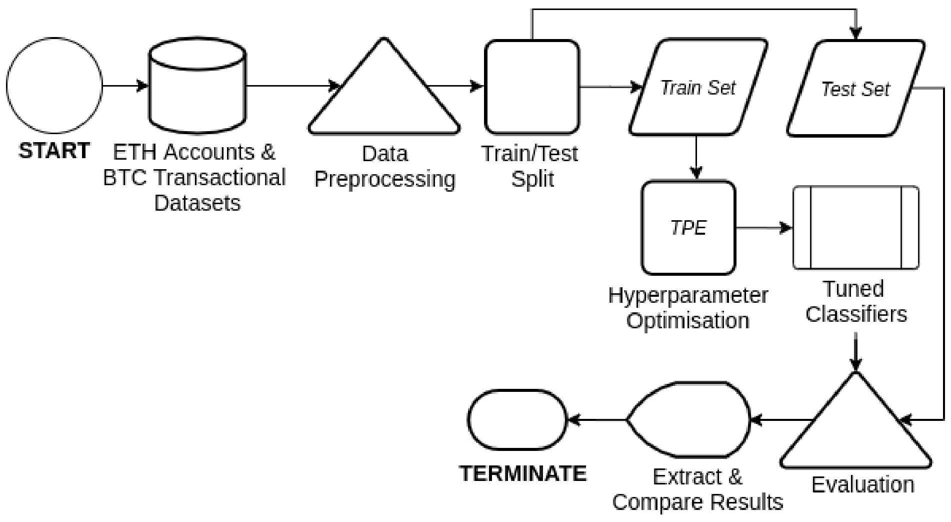


Figure 5.1: Flowchart for Experiment 1.

[4] Orange Python: `https://orange-data-mining-library.readthedocs.io/en/latest/#reference`

5.2.4.2 | Experiment 2

In this experiment, we address hypothesis 2 as defined in Table 4.1. We hypothesise that decision-tree based gradient boosting, in conjunction with data-sampling techniques, can further improve on the classification of licit-or-illicit activities detection at a transactional-level. From this stage onwards, the primary focus shifted towards on transaction-level detection as it permits online detection. In this experiment, we were able to identify whether the implemented data-sampling techniques reduced the False-Negative rate (via Recall measure). A thorough evaluation was also conducted, which helped us to identify the effects of the selected techniques on different models, while also taking note of other performance metrics over Recall (identifying any trade-offs). In this experiment, only the *Elliptic and NOAA datasets* were utilised. The RF model underwent parameter optimisation in this experiment, since the primary transaction-level benchmark dataset was sampled, and the suggested optimal parameters may have not applied on such a sampled dataset.

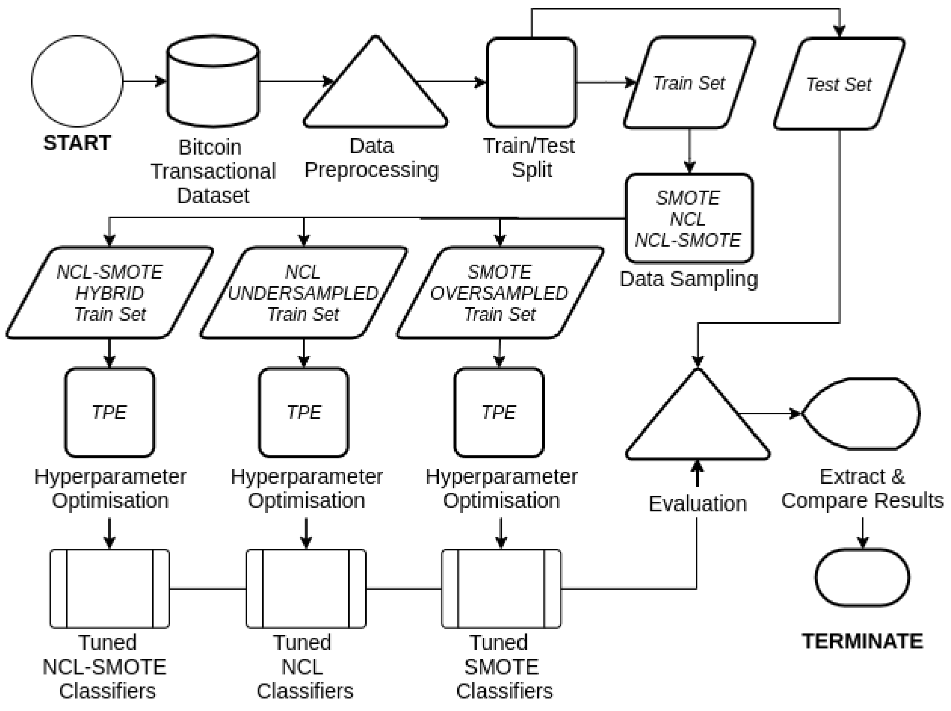

START
Bitcoin Transactional Dataset
Data Preprocessing
Train/Test Split
Train Set
Test Set
SMOTE NCL NCL-SMOTE
Data Sampling
NCL-SMOTE HYBRID Train Set
NCL UNDERSAMPLED Train Set
SMOTE OVERSAMPLED Train Set
TPE
TPE
TPE
Hyperparameter Optimisation
Hyperparameter Optimisation
Hyperparameter Optimisation
Evaluation
Extract & Compare Results
Tuned NCL-SMOTE Classifiers
Tuned NCL Classifiers
Tuned SMOTE Classifiers
TERMINATE

Figure 5.2: Flowchart for Experiment 2.

5.2.4.3 | Experiment 3

In this experiment, we test hypothesis 3 as defined in Table 4.1. We hypothesise that Adaptive Stacked eXtreme Gradient Boosting (ASXGB), developed to handle concept drift, can further improve on the classification of licit-or-illicit transactions, in a stream environment. Our last experiment helped us to evaluate our adaptation of XGBoost to handle non-stationary data streams. More importantly, we were able to compare and contrast against other various techniques, in particular, ARF. As stated, hyperparameter optimisation and data-sampling techniques were not investigated during this experiment due to time constraints.

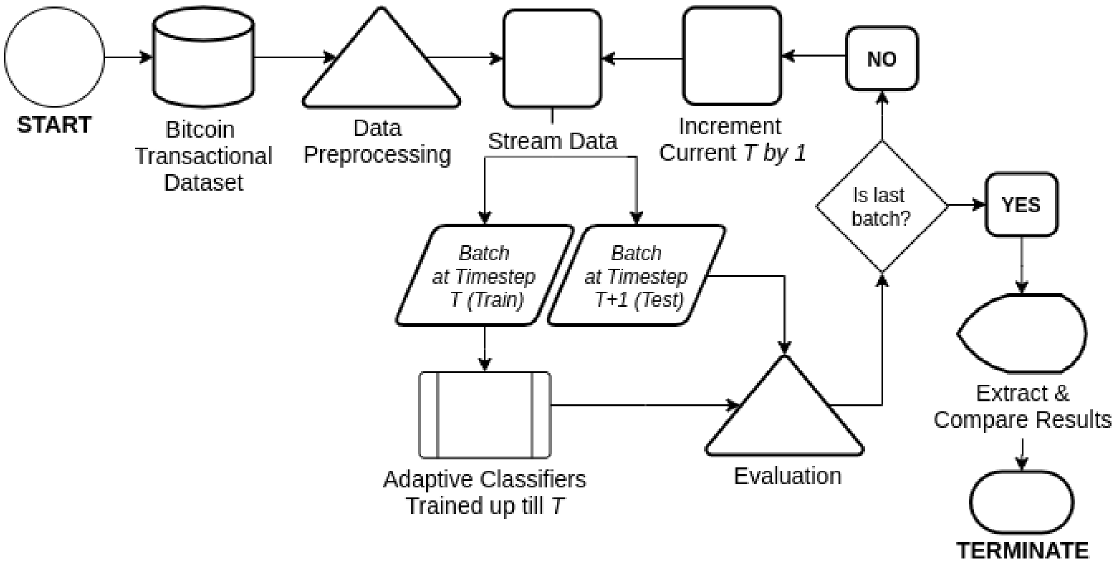

START
Bitcoin Transactional Dataset
Data Preprocessing
Stream Data
Increment Current T by 1
NO
Is last batch?
YES
Batch at Timestep T (Train)
Batch at Timestep T+1 (Test)
Adaptive Classifiers Trained up till T
Evaluation
Extract & Compare Results
TERMINATE

Figure 5.3: Flowchart for Experiment 3.

5.3 | Results for Experiment 1

Inline with Objective 1, this experiment tested whether decision-tree based gradient boosting can improve on the classification of licit-or-illicit activities on both account and transaction level detection, when compared to Random Forest (RF), as referred to in Table 4.1. To highlight the performance efficiency for each evaluated model, Table 5.2 displays the results on the *Elliptic* dataset (Elliptic, 2020; Weber et al., 2019), when evaluating the notion of detecting illicit activities at a transactional-level. The benchmark RF was fitted using the suggested hyperparameters in the study conducted by Weber et al. (2019), to reproduce equivalent results. The gradient boosting algorithms produced relatively better results when hyperparameter tuning was applied, except for the CatBoost algorithm and XGBoost on the *'LF_NE'* feature set, which suffered from a slight degradation of -0.006 in terms of Precision. The optimal hyperparameters found for each tuned model are shown in Listings from B.1 to B.12, in Appendix B. From Table 5.2, it is evident that the *'AF'* and *'AF_NE'* achieved better results across all classifiers, leading to the decision to disregard the other two feature sets, in the upcoming experiments. It can be deduced that the best performing classifier was the tuned $LGBM^{AF}$, as it obtained the highest scores in 3 out of 4 metrics. In order to adequately capture how the results for the best performing models and the benchmark were derived, the confusion matrices for RF, XGBoost, and LGBM on both the *'AF'* and *'AF_NE'*, are shown in Table 5.3. Figures 5.4 and 5.5 display the F1-Score when evaluating on the test set over incrementing timesteps, for the *'AF'* and *'AF_NE'* feature sets, respectively. These figures are displayed in this section so as to be able to compare the outcomes over time with the results obtained in the following experiments.

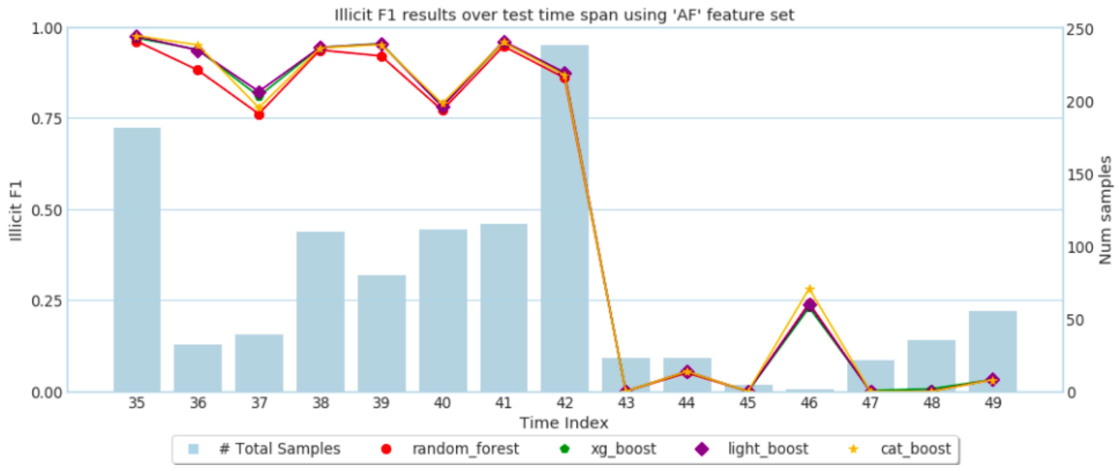


Figure 5.4: F1-Score results indexed by time for the tuned gradient boosting algorithms and the benchmark, over the test set (timestep ≥ 35), when evaluating on the *Elliptic dataset* using the *'AF'* feature set. The results are shown for the following models; RF (Red), XGBoost (Green), LGBM (Violet) and CatBoost (Yellow).

The *Ethereum Illicit Accounts dataset* (Farrugia et al., 2020) was employed to extract the following results, so as to evaluate the concept of detecting illicit activities at an account-level. Table 5.4 shows the performances for each tested model. Since the optimal hyperparameters were still unknown for the benchmark RF model, this underwent hyperparmaeter optimisation. The optimal hyperparameters found during this test are shown in Listings B.13 to B.16, in Appendix B. From the results in Table 5.4, it is apparent that tuning the hyperparameters (shown under *'Tuned Hyperparameters'* in the table),

Exp.1 (*Elliptic dataset*) - Gradient Boosting against Random Forest								
	Default Hyperparameters				Tuned Hyperparameters			
Model	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
XGB^{LF}	0.974	0.877	0.702	0.779	0.976	0.908	0.700	0.790
XGB^{LF_NE}	0.978	**0.988***	0.665	0.795	0.977	0.982	0.665	0.793
XGB^{AF}	0.977	0.902	**0.723**	0.803	0.978	0.921	**0.732***	0.815
XGB^{AF_NE}	**0.979***	0.979	0.693	0.812	**0.979***	**0.986**	0.692	0.813
$LGBM^{LF}$	0.974	0.861	0.711	0.779	0.975	0.888	0.702	0.784
$LGBM^{LF\_NE}$	0.978	0.977	0.681	0.803	0.978	0.984	0.664	0.793
$LGBM^{AF}$	**0.979***	0.931	**0.723**	0.814	**0.979***	0.932	**0.732***	**0.820***
$LGBM^{AF\_NE}$	**0.979***	0.983	0.689	0.810	**0.979***	0.985	0.695	0.815
CAT^{LF}	0.976	0.907	0.701	0.791	0.976	0.892	0.715	0.793
CAT^{LF_NE}	0.978	0.981	0.680	0.803	0.978	0.975	0.672	0.796
CAT^{AF}	**0.979***	0.949	0.721	**0.820***	**0.979***	0.936	0.728	0.819
CAT^{AF_NE}	**0.979***	0.983	0.696	0.815	**0.979***	0.975	0.691	0.809
	Suggested Hyperparameters							
RF^{LF}	0.973	0.868	0.688	0.768	/	/	/	/
RF^{LF_NE}	0.978	0.956	0.697	0.806	/	/	/	/
RF^{AF}	0.977	0.897	0.721	0.800	/	/	/	/
RF^{AF_NE}	**0.979***	0.958	0.715	0.819	/	/	/	/

Table 5.2: Results in this table show the mean performance over 100 runs for the proposed gradient boosting algorithms (XGBoost, LGBM, and CatBoost) and the benchmark RF classifier, extracted in Experiment 1, when evaluated on the *Elliptic* test set (16670 instances, where timestep timestep $\geq$ 35), on all feature sets ('*LF*', '*LF_NE*', '*AF*', and '*AF_NE*'). Outcomes highlighted in bold indicate the best score for a specific section (defaults or tuned hyperparameters), while the ones appended with an '*' indicate the best score from the overall results.

produced better results across all classifiers and almost all metrics, in comparison to the default hyperparameters (shown under *'Default Hyperparameters'* in the table). The tuned XGBoost classifier reigned superior in this test, as it obtained the highest scores in 3 out of all the reported metrics, with the default version obtaining the highest in terms of Precision. Overall, the gradient boosting algorithms outperformed the benchmark RF model, as these algorithms achieved the highest scores for each reported metric. The Confusion matrices shown in Table 5.5, better explain how these reported results (shown in Table 5.4) were derived.

The same procedures employed to evaluate the *Elliptic* and *Ethereum Illicit Accounts* datasets, were implemented on the supplementary *NOAA dataset*. Table 5.6 displays the reported results for this test. Similar to the previous two tests, the models with tuned

	0	1
$0'$	15497 TN	90 FP
$1'$	302 FN	781 TP

(1) RF^{AF}

	0	1
$0'$	15553 TN	34 FP
$1'$	308 FN	775 TP

(2) RF$^{AF\_NE}$

	0	1
$0'$	15519 TN	68 FP
$1'$	291 FN	792 TP

(3) T_XGB^{AF}

	0	1
$0'$	15576 TN	11 FP
$1'$	333 FN	750 TP

(4) T_XGB$^{AF\_NE}$

	0	1
$0'$	15529 TN	58 FP
$1'$	290 FN	793 TP

(5) T_LGBM^{AF}

	0	1
$0'$	15575 TN	12 FP
$1'$	331 FN	752 TP

(6) T_LGBM$^{AF\_NE}$

Table 5.3: These results show the Confusion Matrices for the best performing tuned gradient boosting algorithms and the benchmark RF classifiers in Experiment 1. The results for the *'AF'* and *'AF_NE'* are shown, and each model was trained (29894 instances, where timestep ≤ 34) and tested (16670 instances, where timestep ≥ 35) on the *Elliptic dataset*. In the tables above, columns represent the predicted outcome and rows represent the actual outcome. Models named with a *'T_'* prefix refer to the tuned version.

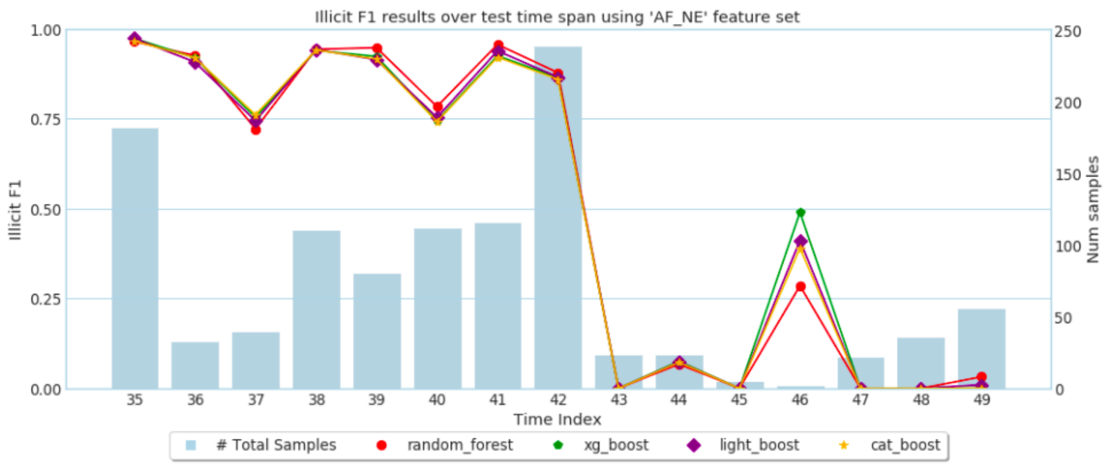


Figure 5.5: F1-Score results indexed by time for the tuned gradient boosting algorithms and the benchmark, over the test set (timestep ≥ 35), when evaluating on the *Elliptic dataset* using the *'AF_NE'* feature set. The results are shown for the following models; RF (Red), XGBoost (Green), LGBM (Violet) and CatBoost (Yellow).

hyperparameters produced relatively better results, except for the CatBoost classifier, as shown in Table 5.6. The optimal hyperparameters found during this test are shown in Listings B.17 to B.20, in Appendix B. The best performing model was the tuned LGBM, as it was able to obtain the highest scores in all metrics. Inline with the previous tests, the gradient boosting models outperformed the benchmark RF model overall, as these models performed the best for all reported metrics. Table 5.7 shows the Confusion matrices, followed by Figure 5.6, which shows the F1-Score for each timestep when evaluating on

Exp.1 (*Ethereum Illicit Accounts dataset*) - Gradient Boosting against Random Forest

	Default Hyperparameters				Tuned Hyperparameters			
Model	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
XGB	**0.981**	**0.989***	0.969	**0.979**	**0.984***	**0.985**	**0.981***	**0.983***
LGBM	0.978	0.984	0.968	0.976	0.981	0.983	0.977	0.980
CAT	0.980	0.980	**0.977**	**0.979**	0.980	0.981	0.977	0.979
RF	0.973	0.982	0.959	0.970	0.974	0.983	0.961	0.972

Table 5.4: Results in this table show the mean performance over 100 runs for the proposed gradient boosting algorithms (XGBoost, LGBM, and CatBoost) and the benchmark RF classifier, extracted in Experiment 1, when evaluated on the *Ethereum Illicit Accounts* test set (1405 instances). Outcomes highlighted in bold indicate the best score for a specific section (defaults or tuned hyperparameters), while the ones appended with an ′*′ indicate the best score from the overall results.

	0	1
$0'$	739 *TN*	12 *FP*
$1'$	27 *FN*	627 *TP*

(1) RF

	0	1
$0'$	740 *TN*	11 *FP*
$1'$	26 *FN*	628 *TP*

(2) T_RF

	0	1
$0'$	744 *TN*	7 *FP*
$1'$	20 *FN*	634 *TP*

(3) XGB

	0	1
$0'$	741 *TN*	10 *FP*
$1'$	12 *FN*	642 *TP*

(4) T_XGB

	0	1
$0'$	741 *TN*	10 *FP*
$1'$	21 *FN*	633 *TP*

(5) LGBM

	0	1
$0'$	740 *TN*	11 *FP*
$1'$	15 *FN*	639 *TP*

(6) T_LGBM

	0	1
$0'$	738 *TN*	13 *FP*
$1'$	15 *FN*	639 *TP*

(7) CAT

	0	1
$0'$	739 *TN*	12 *FP*
$1'$	15 *FN*	639 *TP*

(8) T_CAT

Table 5.5: These results show the Confusion Matrices for the gradient boosting algorithms and the benchmark RF classifiers in Experiment 1. Each model was trained (3276 instances) and tested (1405 instances) on the *Ethereum Illicit Accounts dataset*. In the tables above, columns represent the predicted outcome and rows represent the actual outcome. Models named with a ′*T_*′ prefix refer to the tuned version.

the test-set.

Lastly, for this experiment, several statistical tests were applied to identify any statistical differences between the reported results. Box-plots were plotted for the Precision, Recall, and F1-Score results obtained when evaluating on multiple datasets. These box plots helped us to visually identify the most stable models (less variation in the reported results) when taking into consideration that each model was ran for 100 times.

Exp.1 (*NOAA weather dataset*) - Gradient Boosting against Random Forest								
	Default Hyperparameters				Tuned Hyperparameters			
Model	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
XGB	0.776	0.729	0.594	0.655	0.785	0.749	0.596	0.664
LGBM	0.784	0.735	0.617	0.671	**0.795***	**0.750***	**0.639***	**0.690***
CAT	**0.792**	**0.748**	**0.630**	**0.684**	0.790	0.744	0.629	0.682
RF	0.78	0.742	0.589	0.657	0.784	0.748	0.595	0.662

Table 5.6: Results in this table show the mean performance over 100 runs for the proposed gradient boosting algorithms (XGBoost, LGBM, and CatBoost) and the benchmark RF classifier, extracted in Experiment 1, when evaluated on the *NOAA* test set (5439 instances, where timestep $\geq$ 425). Outcomes highlighted in bold indicate the best score for a specific section (defaults or tuned hyperparameters), while the ones appended with an ′*′ indicate the best score from the overall results.

	0	1
$0'$	3100 *TN*	397 *FP*
$1'$	799 *FN*	1143 *TP*

(1) RF

	0	1
$0'$	3107 *TN*	390 *FP*
$1'$	787 *FN*	1155 *TP*

(2) T_RF

	0	1
$0'$	3069 *TN*	428 *FP*
$1'$	789 *FN*	1153 *TP*

(3) XGBoost

	0	1
$0'$	3109 *TN*	388 *FP*
$1'$	784 *FN*	1158 *TP*

(4) T_XGBoost

	0	1
$0'$	3065 *TN*	432 *FP*
$1'$	744 *FN*	1198 *TP*

(5) LGBM

	0	1
$0'$	3083 *TN*	414 *FP*
$1'$	700 *FN*	1242 *TP*

(6) T_LGBM

	0	1
$0'$	3084 *TN*	413 *FP*
$1'$	719 *FN*	1223 *TP*

(7) CatBoost

	0	1
$0'$	3077 *TN*	420 *FP*
$1'$	721 *FN*	1221 *TP*

(8) T_CatBoost

Table 5.7: These results show the Confusion Matrices for the gradient boosting algorithms and the benchmark RF classifiers in Experiment 1. Each model was trained (12720 instances, where timestep $\leq$ 424) and tested (5439 instances, where timestep $\geq$ 425) on the *NOAA dataset*. In the tables above, columns represent the predicted outcome and rows represent the actual outcome. Models named with a ′*T_*′ prefix refer to the tuned version.

Figures C.1 to C.4 in Appendix C, show the box-plots for the *Elliptic dataset* where LGBM is the most stable model for almost all the reported metrics across all tested feature sets. Figure C.5 in Appendix C, show the box-plot for the *Ethereum Illicit Accounts dataset*, where XGBoost is the most stable for almost all the reported metrics. Figures C.6 in Appendix C, show the box-plot for the supplementary *NOAA dataset*, where LGBM is

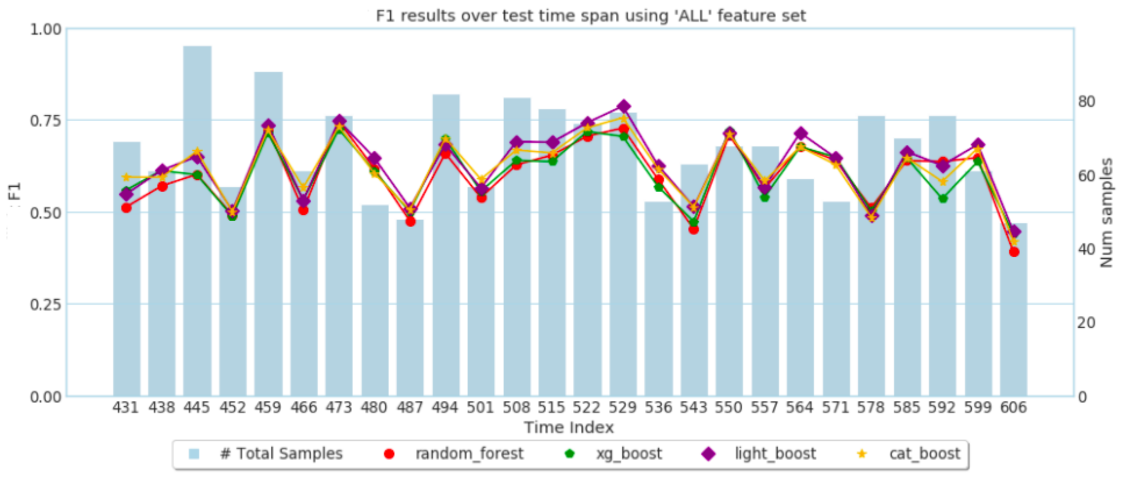


Figure 5.6: F1-Score indexed by time, when evaluating the proposed and benchmark models on the *NOAA* test set (5439 instances, where timestep $\geq$ 425). Each column in this bar chart represent 7 timesteps, so the first column represents 425 to 431, the second bar represents 432 to 438, and so on. The results are shown for the following models; RF (Red), XGBoost (Green), LGBM (Violet) and CatBoost (Yellow).

the most stable for almost all the reported metrics. To test if there were any statistical differences between the models, when evaluated on the *Elliptic* (transactional-level detection) and the *Ethereum Illicit Accounts* (account-level detection) datasets, a non-parametric version of the t-test was employed. The decision to apply a non-parametric hypothesis test over a parametric one, was taken due to the results (over a 100 runs) not being normally distributed. The scipy.stats.normaltest was applied to test for normality, based on the method described by Pearson et al. (1977). The Wilcoxon signed-rank test was employed to compare whether two classifiers have the same performance power statistically, or if one is better than the other. The null hypothesis states that the median difference is equal to 0 (same performance). For the *Elliptic dataset*, we compared the distribution which produces the highest mean score obtained by the benchmark, against the distribution that produced the highest mean score, obtained by the proposed tuned gradient boosting algorithms. Table 5.8 shows the results obtained during this test.

The same procedure was applied to the reported results obtained when evaluating on the *Ethereum Illicit Accounts dataset*, and the outcomes are shown in Table 5.9.

Furthermore, statistical significance for Recall, Precision, and F1-Score between classifiers over multiple datasets - *Elliptic* with *LF*, *LF_NE*, *AF*, and *AF_NE* feature sets, along with *Ethereum Illicit Accounts dataset* was also tested. The Friedman Chi-Squared test was conducted on the following metrics; Precision, Recall, and F1-Score, which outputted the following p-values; 0.037, 0.706, and 0.642, respectively. It was established,

Boosting (Mean Score)	Benchmark (Mean Score)	Metric	Outcome (p-value < 0.05)
$T\_XGB^{AF\_NE}$ (0.986)	RF^{AF_NE} (0.958)	Precision	reject H_0
T_XGB^{AF} (0.732)	RF^{AF} (0.721)	Recall	reject H_0
T_LGBM^{AF} (0.732)	RF^{AF} (0.721)	Recall	reject H_0
T_LGBM^{AF} (0.820)	RF^{AF_NE} (0.819)	F1-Score	reject H_0

Table 5.8: These results were obtained when applying the Wilcoxon signed-rank test (significance level of 0.05), in order to check for statistical differences between the reported results during the *Elliptic dataset* (transactional-level detection) evaluation. Models named with a *'T_'* prefix refer to the tuned version.

Boosting (Mean Score)	Benchmark (Mean Score)	Metric	Outcome (p-value < 0.05)
T_XGB (0.985)	T_RF (0.983)	Precision	reject H_0
T_XGB (0.981)	T_RF (0.961)	Recall	reject H_0
T_XGB (0.983)	T_RF (0.972)	F1-Score	reject H_0

Table 5.9: These results were obtained when applying the Wilcoxon signed-rank test (significance level of 0.05), in order to check for statistical differences between the reported results during the *Ethereum Illicit Accounts dataset* (account-level detection) evaluation. Models named with a *'T_'* prefix refer to the tuned version.

that at least two of the tested classifiers obtained Precision scores which were statistically different (0.037 $\leq$ 0.05). The Nemenyi post-hoc test was then carried out, so as to identify which classifiers across all datasets, are significant from each other in terms of Precision . The resulting outcome is shown in the Nemenyi Critical Difference diagram, in Figure 5.7. Since the average ranks for both the tuned XGBoost and LGBM differ from the critical difference (1.91), as shown in this figure, these two models are statistically different from the RF benchmark.

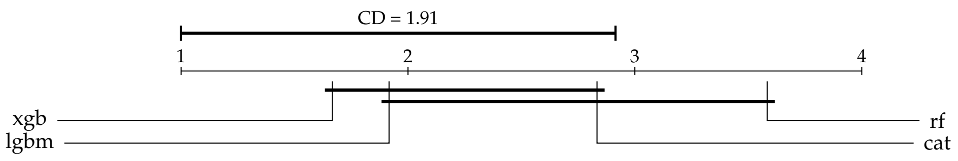


Figure 5.7: Nemenyi Post-Hoc test when evaluating statistical difference for Precision in Experiment 1. Models which are not connected to each other, are statistically different ($\alpha = 0.05$)

The CatBoost classifier was dropped for the upcoming experiments, as it was outperformed by the other two gradient boosting algorithms on all the tested datasets, as shown in Tables 5.2, 5.4, and 5.6.

5.4 | Results for Experiment 2

Inline with Objective 2, this experiment tested whether decision-tree based gradient boosting, in conjunction with data-sampling techniques, can further improve on the classification of licit-or-illicit activities detection at a transactional-level, as referred to in Table 4.1. In this analysis the following data-sampling techniques; Neighbourhood Cleaning Rule (NCL), Synthetic Minority Over-Sampling (SMOTE), and NCL-SMOTE were applied on both the *Elliptic* and *NOAA* datasets.

Since NCL, SMOTE, and NCL-SMOTE were used to sample the *Elliptic dataset*, the number of instances utilised during the training phase changed by; -1203, $+22970$, and $+20564$, respectively. Table 5.10, shows the results obtained by the tested models on the sampled *Elliptic dataset*. Contrary to the first experiment, hyperparameter optimisation was applied to the benchmark RF classifier, as the optimal hyperparameters were still unknown for the sampled *Elliptic dataset*. Overall, hyperparameter optimisation produced better results, as shown in Table 5.10; however, it is evident that the Recall score seems to degrade at the expense of improving the Precision in some classifiers, most notably, when applied to the $NCL\_SMOTE\_XGB^{AF}$ model, where Recall degraded by -0.8%. The optimal hyperparameters found during this test are shown in Listings B.21 to B.38, in Appendix B. The highest scores for Accuracy, Precision, Recall, and F1-Score, all of which are shown in Table 5.10, were obtained by; RF^{AF_NE} in conjunction with NCL, tuned XGB^{AF_NE} in conjunction with NCL, XGB^{AF} in conjunction with NCL-SMOTE, and RF^{AF_NE} in conjunction with NCL, respectively. Comparing these results with those in Table 5.2 (Experiment 1), it is evident that overall the RF classifier seemed to benefit the most from the use of Data-Sampling techniques. Nonetheless, there were slight improvements when these techniques were employed with the proposed gradient boosting algorithms, especially the 1% improvement in Recall. This increase is significant since the Confusion matrices displayed in Table 5.3 in Experiment 1, seem to indicate that there is a high level of False Negatives, with the lowest being 290. To compare and contrast against the previous experiment, the Confusion matrices for the models which obtained the top six Recall scores, were plotted and shown in Table 5.11. Five out of six models in this table obtained a lower False Negatives count than the lowest score obtained in Experiment 1, with the $NCL-SMOTE\_XGB^{AF}$ model having the highest difference, with 11 less False Negatives. Lastly, Figure 5.8 shows the F1-Score over the test time span when NCL was applied (selected due to it being the Data-Sampling technique which obtained the highest F1-Score). When comparing this figure with Figures 5.4 and 5.5 in Experiment 1, it can be noted that there were very minimal improvements, and the problem of non-stationarity is still impeding performance.

Exp.2 (Sampled *Elliptic dataset*) - Gradient Boosting against Random Forest								
	Default Hyperparameters				Tuned Hyperparameters			
Model	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
NCL_XGB^{AF}	0.977	0.911	0.724	0.807	0.978	0.913	0.734	0.814
$NCL\_XGB^{AF\_NE}$	0.979	0.966	0.699	0.811	0.979	**0.985***	0.687	0.809
$SMOTE\_XGB^{AF}$	0.978	0.904	0.731	0.809	0.980	0.939	**0.735**	0.824
$SMOTE\_XGB^{AF\_NE}$	0.980	**0.976**	0.704	0.818	0.980	0.975	0.716	0.826
$NCL\_SMOTE\_XGB^{AF}$	0.976	0.875	**0.742***	0.803	0.979	0.924	0.734	0.818
$NCL\_SMOTE\_XGB^{AF\_NE}$	0.979	0.945	0.711	0.811	0.980	0.965	0.721	0.825
NCL_LGBM^{AF}	0.977	0.905	0.724	0.805	0.978	0.918	0.732	0.814
$NCL\_LGBM^{AF\_NE}$	0.979	0.974	0.700	0.815	0.979	0.980	0.688	0.809
$SMOTE\_LGBM^{AF}$	0.976	0.883	0.735	0.802	0.980	0.939	0.733	0.823
$SMOTE\_LGBM^{AF\_NE}$	0.979	0.955	0.711	0.815	0.980	0.980	0.705	0.820
$NCL\_SMOTE\_LGBM^{AF}$	0.976	0.863	0.741	0.797	0.980	0.942	0.732	0.824
$NCL\_SMOTE\_LGBM^{AF\_NE}$	0.979	0.938	0.721	0.815	0.980	0.972	0.708	0.819
NCL_RF^{AF}	0.979	0.940	0.725	0.819	0.979	0.945	0.726	0.821
$NCL\_RF^{AF\_NE}$	**0.981***	0.973	0.721	**0.828**	**0.981***	0.977	0.723	**0.831***
$SMOTE\_RF^{AF}$	0.978	0.922	0.720	0.808	0.978	0.929	0.721	0.812
$SMOTE\_RF^{AF\_NE}$	0.979	0.955	0.717	0.819	0.980	0.965	0.722	0.826
$NCL\_SMOTE\_RF^{AF}$	0.975	0.875	0.723	0.792	0.976	0.878	0.725	0.794
$NCL\_SMOTE\_RF^{AF\_NE}$	0.979	0.933	0.723	0.815	0.979	0.945	0.726	0.821

Table 5.10: Results in this table show the mean performance over 100 runs for the proposed gradient boosting algorithms (*XGB* and *LGBM*) and the benchmark RF classifier, extracted in Experiment 2 when evaluated on the *Elliptic* test set (timestep $\geq$ 35), on two different feature sets - '*AF*', '*AF_NE*'. All the evaluated models were fitted on three different sampled training sets (timestep $\leq$ 34) using the following Data-Sampling techniques; NCL, SMOTE, and NCL-SMOTE. Outcomes highlighted in bold indicate the best score for a specific section (defaults or tuned hyperparameters), while the ones highlighted with '*' indicate the best score from the overall results.

The same procedures were also applied on the supplementary *NOAA* dataset. Table 5.12 shows the performances reported during this test. The optimal hyperparameters found during this analysis are shown in Listings B.39 to B.47 in Appendix B. Similarly to the previous test, the Recall score is the primary improvement when employing these Data-Sampling techniques, with a difference of 0.215 (obtained by *NCL_LGBM*) when compared with the highest score in Experiment 1. Although the Recall score improved, the highest Precision score obtained degraded by 0.039, when compared to Experiment 1. An improvement in the F1-Score was also seen in comparison to first experiment (+0.013). Comparing the F1-Score over time, as shown in Figure 5.9 (showing only NCL, due to it being the technique which obtained the highest F1-Score) with Figure 5.6 in Experiment 1, it can be seen that F1-Score is oscillating slightly higher than the previous results.

	0	1
0′	15472 TN	115 FP
1′	279 FN	804 TP

(1) NCL-SMOTE_XGB^{AF}

	0	1
0′	15460 TN	127 FP
1′	281 FN	802 TP

(2) NCL-SMOTE_LGBM^{AF}

	0	1
0′	15482 TN	105 FP
1′	287 FN	796 TP

(3) SMOTE_LGBM^{AF}

	0	1
0′	15535 TN	52 FP
1′	287 FN	796 TP

(4) T_SMOTE_XGB^{AF}

	0	1
0′	15511 TN	76 FP
1′	288 FN	795 TP

(5) T_NCL_XGB^{AF}

	0	1
0′	15516 TN	71 FP
1′	291 FN	792 TP

(6) T_NCL_LGBM^{AF}

Table 5.11: Confusion matrices results for the models with lowest False Negatives when evaluated on the sampled *Elliptic dataset*. Models named with a *′T_′* prefix refer to the tuned version.

Exp.2 (Sampled *NOAA dataset*) - Gradient Boosting against Random Forest								
	Default Hyperparameters				Tuned Hyperparameters			
Model	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
NCL_XGB	0.743	0.602	0.827	**0.697**	0.746	0.606	0.829	0.700
SMOTE_XGB	**0.779**	**0.706**	0.654	0.679	**0.781***	**0.711***	0.652	0.680
NCL_SMOTE_XGB	0.753	0.620	0.795	0.696	0.731	0.585	**0.843**	0.691
NCL_LGBM	0.733	0.587	**0.854***	0.696	0.752	0.613	0.824	**0.703***
SMOTE_LGBM	0.776	0.684	0.690	0.687	0.778	0.708	0.646	0.675
NCL_SMOTE_LGBM	0.736	0.593	0.832	0.692	0.736	0.593	0.834	0.693
NCL_RF	0.746	0.610	0.799	0.692	0.746	0.609	0.803	0.693
SMOTE_RF	0.771	0.679	0.679	0.679	0.771	0.677	0.686	0.682
NCL_SMOTE_RF	0.734	0.593	0.816	0.687	0.733	0.591	0.820	0.687

Table 5.12: Results in this table show the mean performance over 100 runs for the proposed gradient boosting algorithms (*XGB* and *LGBM*) and the benchmark RF classifier, extracted in Experiment 2 when evaluated on the *NOAA* test set (timestep $\geq$ 425). All the evaluated models were fitted on three different sampled training sets (timestep $\leq$ 424) using the following Data-Sampling techniques; NCL, SMOTE, and NCL-SMOTE. Outcomes highlighted in bold indicate the best score for a specific section (defaults or tuned hyperparameters), while the ones highlighted with ′*′ indicate the best score from the overall results.

The distribution of results for both the *Elliptic* and *NOAA* datasets, are shown in the form of box-plots in Figures C.7 to C.12 and Figures C.13 to C.15 in Appendix C, respectively. From these box-plots LGBM showed to be the most stable relative to the

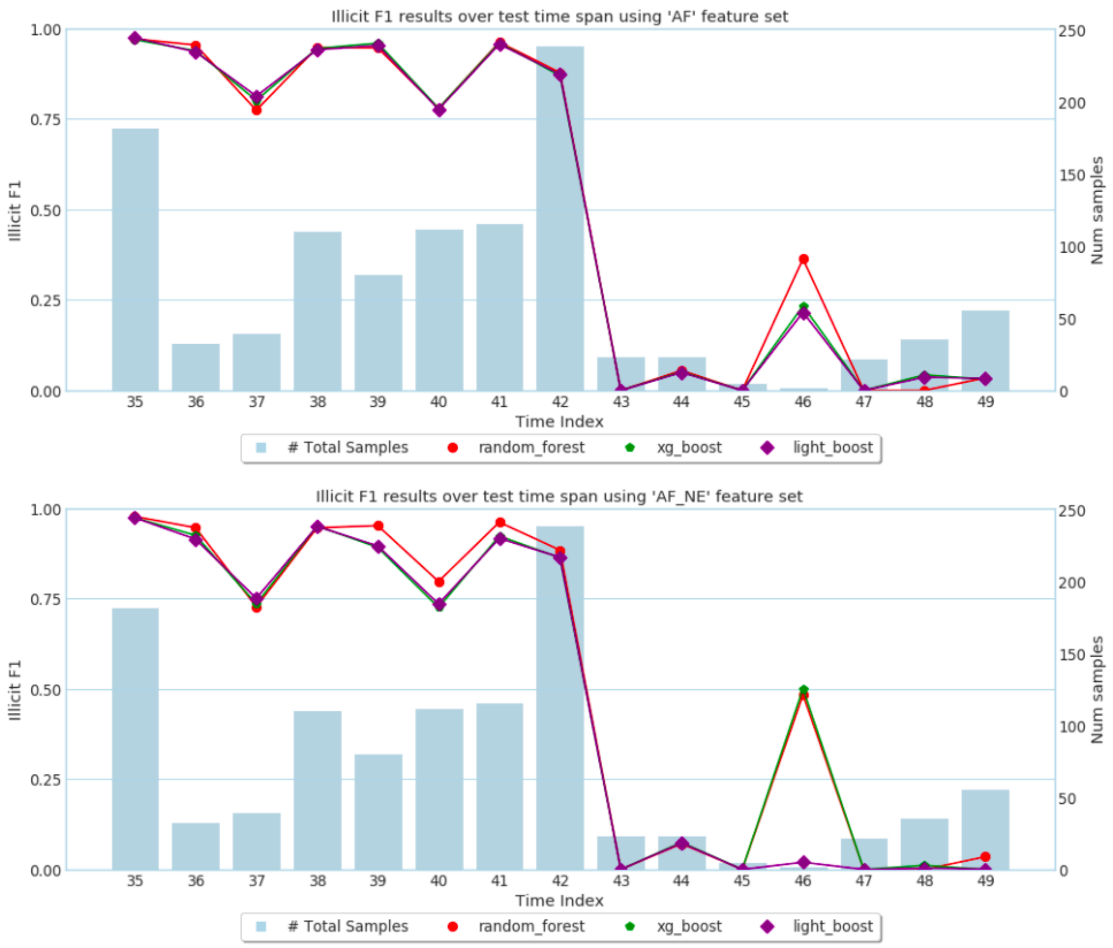


Figure 5.8: F1-Score results indexed by time for the tuned gradient boosting algorithms and the tuned benchmark, over the test set (timestep ≥ 35), when trained on a sampled *Elliptic* training set (28691 instances where timestep ≤ 34) using NCL and evaluated on the *Elliptic dataset* using the *'AF'* (Top) and *'AF_NE'* (Bottom) feature sets. The results are shown for the following models; RF (Red), XGBoost (Green) and LGBM (Violet).

other tested models, across all data-sampling methods, metrics and feature sets when evaluating on the *Elliptic* dataset. The Wilcoxon signed-rank test, was utilised to test for statistical significance, so as to prove whether Data-Sampling techniques improve performance when applied to the *Elliptic dataset*. In order to do so, the tuned XGBoost model that obtained the highest Recall in Experiment 1, was compared with the tuned XGBoost with the highest Recall from this experiment. This procedure was also conducted for the LGBM and RF classifiers. Table 5.13 shows the outcomes produced during this hypothesis test.

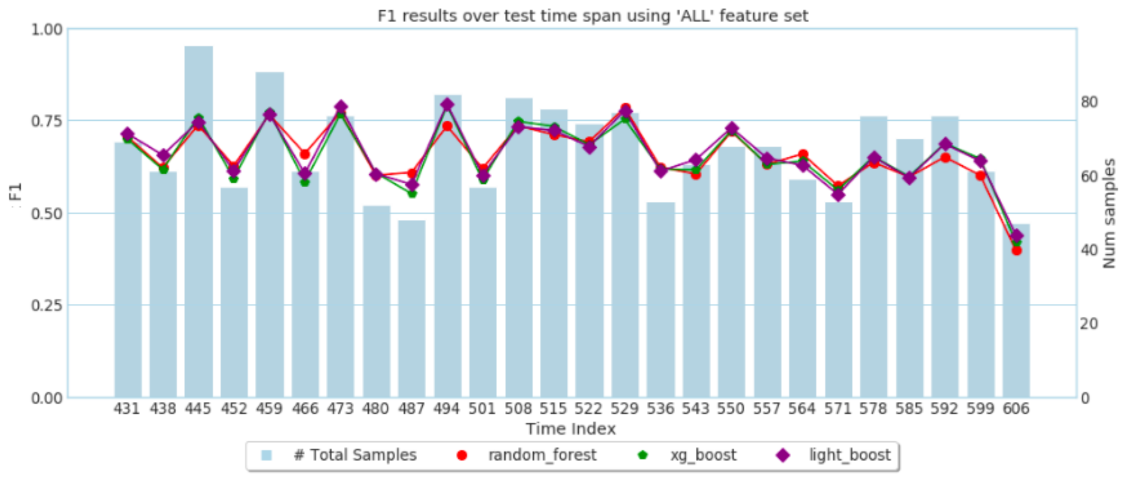


Figure 5.9: F1-Score results indexed by time for the tuned gradient boosting algorithms and the tuned benchmark, over the test set (timestep ≥ 425), when trained on a sampled *NOAA* training set (9229 instances where timestep ≤ 424) using NCL. The results are shown for the following models; RF (Red), XGBoost (Green) and LGBM (Violet).

With Sampling (Mean Score)	No Sampling (Mean Score)	Metric	Outcome (p-value < 0.05)
XGB			
$T\_SMOTE\_XGB^{AF}$ (0.939)	T_XGB^{AF} (0.921)	Precision	reject H_0
$T\_SMOTE\_XGB^{AF}$ (0.735)	T_XGB^{AF} (0.732)	Recall	reject H_0
$T\_SMOTE\_XGB^{AF}$ (0.824)	T_XGB^{AF} (0.815)	F1-Score	reject H_0
LGBM			
$T\_SMOTE\_LGBM^{AF}$ (0.939)	T_LGBM^{AF} (0.932)	Precision	reject H_0
$T\_SMOTE\_LGBM^{AF}$ (0.733)	T_LGBM^{AF} (0.732)	Recall	reject H_0
$T\_SMOTE\_LGBM^{AF}$ (0.823)	T_LGBM^{AF} (0.820)	F1-Score	reject H_0
RF			
$T\_NCL\_RF^{AF}$ (0.945)	RF^{AF} (0.897)	Precision	reject H_0
$T\_NCL\_RF^{AF}$ (0.726)	RF^{AF} (0.721)	Recall	reject H_0
$T\_NCL\_RF^{AF}$ (0.821)	RF^{AF} (0.800)	F1-Score	reject H_0

Table 5.13: These results were obtained when applying the Wilcoxon signed-rank test (significance level of 0.05), in order to check for statistical differences between the reported results during Experiment 1 (without Data-Sampling) against Experiment 2 (with Data-Sampling) when evaluating on the *Elliptic dataset* (transactional-level detection). Models named with a *'T_'* prefix refer to the tuned version.

5.5 | Results for Experiment 3

Inline with Objective 3, this experiment tested whether Adaptive Stacked eXtreme Gradient Boosting (ASXGB), developed to handle concept drift, can further improve on the classification of licit-or-illicit transactions, in a stream environment, as referred to in Table 4.1. The use of various adaptive learners on both the *Elliptic* (transactional-level detection) and the supplementary *NOAA* datasets were tested. In particular, the following models were evaluated; Adaptive Random Forest (ARF), Adaptive eXtreme Gradient Boosting (AXGB) with 'replacement' updates, AXGB with 'push' updates, and our own proposed adaptation of XGBoost, called Adaptive Stacked eXtreme Gradient Boosting (ASXGB). The prequential evaluation was employed during this test, meaning that the models would be trained on timestep t_i and evaluated on t_{i+1}, until $i = T$ (total number of timesteps). Table 5.14 shows the results reported when evaluating on the *Elliptic dataset*, on both the '*AF*' and '*AF_NE*' feature sets. The suggested hyperparameters in the study conducted by Montiel et al. (2020), were used for AXGB, while the default hyperparameters for ARF were used. For the ASXGB implementation, manual hyperparameter tuning was employed, which resulted in the following configuration; maximum window size was set to 2000, number of base learners was set to 5, train ratio for the meta learner was set to 0.4, and the number of rounds used for evaluating the base models was set to 5. In order to compare the results against the previous two experiments, the models were evaluated on timesteps ≥ 5 and timesteps ≥ 35 (inline with the previous experiments). It is clear from these results, that ARF outperformed all the other adaptive learners, since it achieved the highest score in 3 of all the reported metrics on both the $t \geq 35$ and $t \geq 5$ evaluations. Nonetheless, the proposed ASXGB obtained the highest Recall scores in both evaluations, which was very crucial when evaluating on the *Elliptic dataset*, as the main bottleneck from the previous experiments was the high count of False Negatives. In fact, the $ASXGB^{AF}$ obtained a total of 251 False Negatives when evaluated on $t \geq 35$. This was 28 less from Experiment 2, and 39 less from Experiment 1, when compared with the previously reported lowest False Negatives count. It is worth noting, that even though our ASXGB ranked second in terms of F1-Score (both $t \geq 35$ and $t \geq 5$ evaluations), this model shows to be the fastest to adapt to changes (in distribution), as shown in Figures 5.10, 5.11, 5.12 and 5.13, after timestep 43. Comparing Figures 5.10 and 5.11, with the previous F1-Score time plots from the other experiments, it is evident that these adaptive learners are more effective in adjusting to distributional changes over time.

These procedures were also applied on the supplementary *NOAA* dataset, and below are the results reported during the prequential evaluation. The evaluation was only executed on $t \geq 25$ (due to time constraints $t \geq 425$ was not included) and the AXGB (with replacement update) was not tested, as it was the worst-performing in terms of F1-Score in the evaluation conducted on the *Elliptic dataset* (Experiment 3 where $t \geq 5$). Table 5.15 shows the results reported during this test. Similar to the previous evaluation

Exp.3 (*Elliptic dataset*) - Prequential Evaluation: Adaptive Learners								
	timestep $\geq$ 35				timestep $\geq$ 5			
Model	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
ARF^{AF}	**0.977**	**0.988**	0.657	**0.789**	**0.969**	**0.986**	0.732	**0.840**
ARF^{AF_NE}	**0.977**	0.987	0.648	0.783	0.968	0.979	0.724	0.832
$AXGB[R]^{AF}$	0.949	0.59	0.719	0.648	0.947	0.813	0.68	0.74
$AXGB[R]^{AF\_NE}$	0.962	0.713	0.694	0.704	0.954	0.871	0.688	0.769
$AXGB[P]AF$	0.947	0.572	0.722	0.639	0.948	0.778	0.738	0.757
$AXGB[P]^{AF\_NE}$	0.953	0.628	0.684	0.655	0.952	0.792	0.765	0.778
$ASXGB^{AF}$	0.961	0.674	**0.768**	0.718	0.961	0.828	0.817	0.822
$ASXGB^{AF\_NE}$	0.958	0.663	0.728	0.694	0.96	0.813	**0.831**	0.822

Table 5.14: Results in this table show the performance obtained by the evaluated adaptive learners - ARF, AXGB with 'replacement' updates ($AXGB[R]$), AXGB with 'push' updates ($AXGB[P]$), and the proposed ASXGB on the *Elliptic dataset* using both '*AF*' and '*AF_NE*' feature sets. The results on the left show the reported scores when the models were evaluated after $t \geq 35$, and on the right after $t \geq 5$. Outcomes highlighted in bold indicate the best score for a specific section.

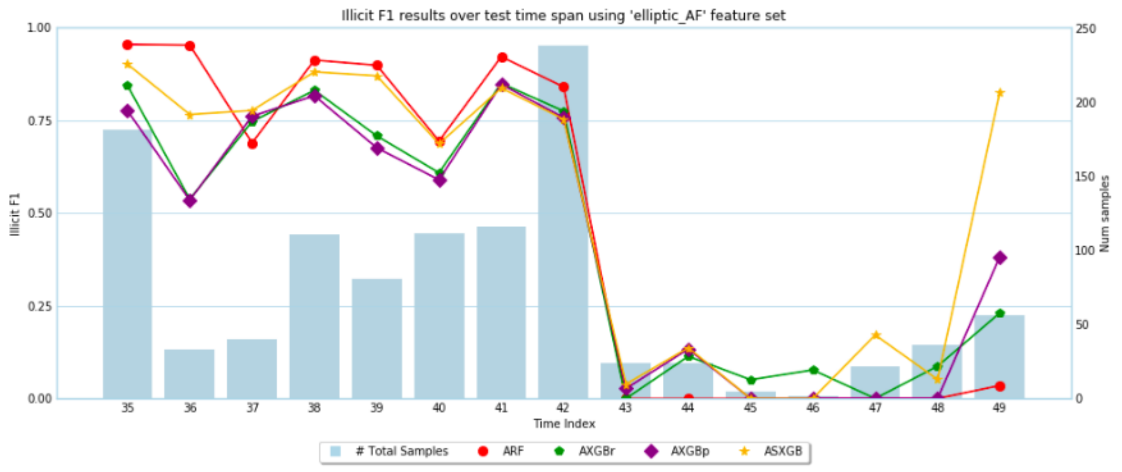


Figure 5.10: F1-Score results indexed by time for the evaluated adaptive learners, when evaluating on the *Elliptic dataset* on '*AF*' feature set, with $t \geq 35$. The results are shown for the following models; ARF (Red), AXGB with replacement (Green), AXGB with push (Violet), and the proposed ASXGB (Yellow).

(*Elliptic* - Experiment 3), our proposed ASXGB obtained the highest score in terms of Recall, however, in this test it ranked last in terms of F1-Score as shown in Table 5.15 and Figure 5.14, which can be attributed with the low Precision result. Even though the evaluation in this test started at $t \geq 25$ (in order to simulate streaming data and utilise the

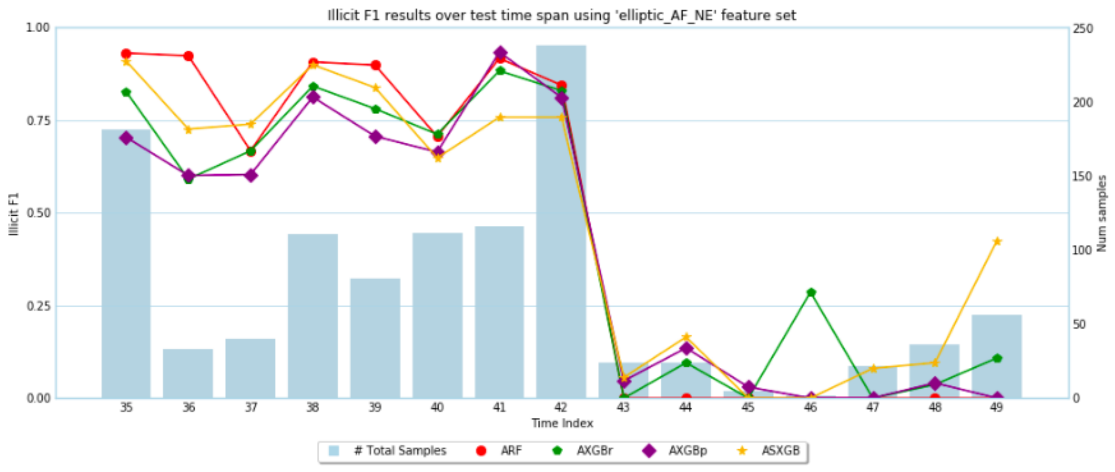


Figure 5.11: F1-Score results indexed by time for the evaluated adaptive learners, when evaluating on the *Elliptic dataset* on '*AF_NE*' feature set, with $t \geq 35$. The results are shown for the following models; ARF (Red), AXGB with replacement (Green), AXGB with push (Violet), and the proposed ASXGB (Yellow).

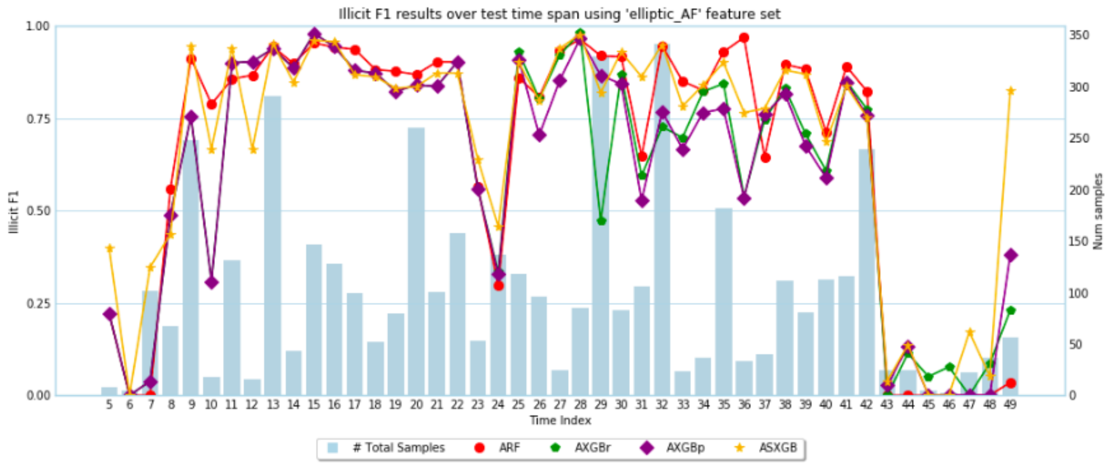


Figure 5.12: F1-Score results indexed by time for the evaluated adaptive learners, when evaluating on the *Elliptic dataset* on '*AF*' feature set, with $t \geq 5$. The results are shown for the following models; ARF (Red), AXGB with replacement (Green), AXGB with push (Violet), and the proposed ASXGB (Yellow).

full dataset), it is evident that overall the adaptive learners produced relatively lower results, when comparing previous experiments where the evaluation started at $t \geq 425$.

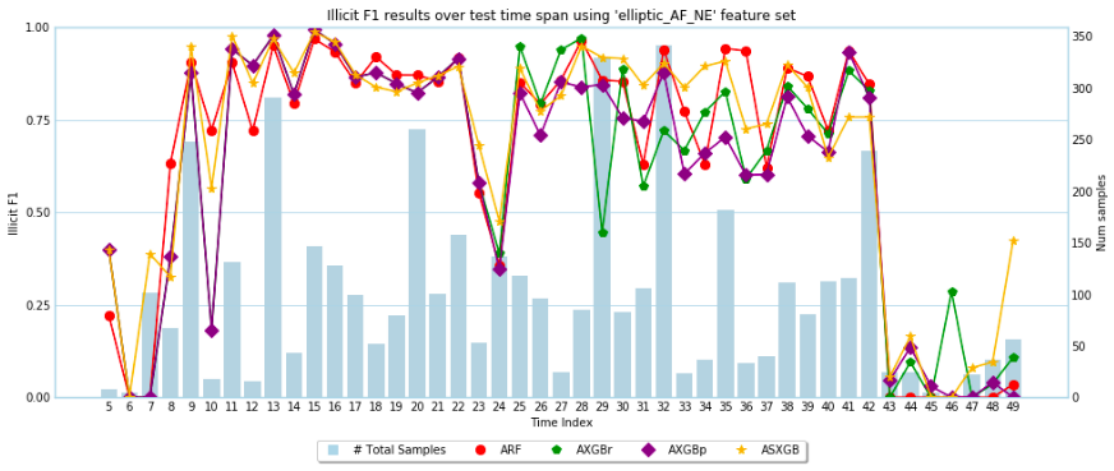


Figure 5.13: F1-Score results indexed by time for the evaluated adaptive learners, when evaluating on the *Elliptic dataset* on *'AF_NE'* feature set, with $t \geq 5$. The results are shown for the following models; ARF (Red), AXGB with replacement (Green), AXGB with push (Violet), and the proposed ASXGB (Yellow).

Exp.3 (*NOAA dataset*) - Prequential Evaluation: Adaptive Learners				
	timestep ≥ 25			
Model	Accuracy	Precision	Recall	F1
ARF	**0.780**	**0.713**	0.501	0.589
AXGB[P]	0.777	0.690	0.528	**0.598**
ASXGB	0.696	0.516	**0.570**	0.542

Table 5.15: Results in this table show the performance obtained by the evaluated adaptive learners - ARF, AXGB with 'push' updates ($AXGB[P]$), and the proposed ASXGB on the *NOAA dataset*. The results show the reported scores when the models were evaluated after $t \geq 25$. Outcomes highlighted in bold indicate the best score for a specific section.

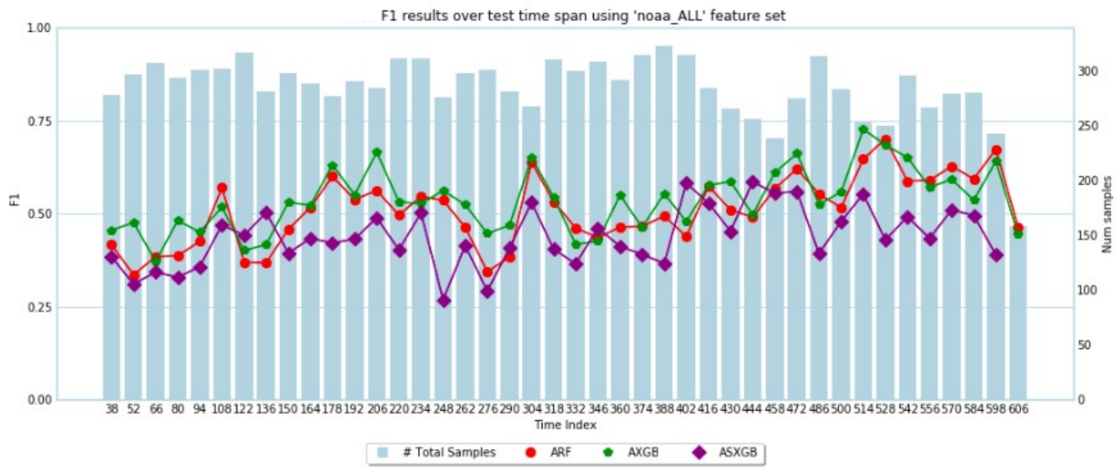


Figure 5.14: F1-Score results indexed by time for the evaluated adaptive learners, when evaluating on the *NOAA dataset*, with $t \geq 25$. The results are shown for the following models; ARF (Red), AXGB with push (Green), AXGB with push (Violet), and the proposed ASXGB (Yellow).

6

Discussion

6.1 | Improved performance using XGBoost & LGBM at an Account and Transactional-Level

The results in Experiment 1 showed that the proposed gradient boosting algorithms (XGBoost and LGBM) outperformed the benchmark RF classifier in identifying illicit activities on the blockchain, both at a transactional-level (*Elliptic*) and at an account level (*Ethereum Illicit Accounts*). With further testing, it was shown that these results (Table 5.2 and Table 5.4) had statistical significance (Tables 5.8 and 5.9). To our disappointment, the CatBoost classifier did not produce results on par with those produced by the XGBoost and LGBM classifiers, as the expectation was that this algorithm would at least obtain similar results to these two models, as it was shown to outperform them in a study conducted by Prokhorenkova et al. (2018). It was also shown that when evaluating on the *Elliptic dataset*, the *'AF'* and *'AF_NE'* feature sets produced better results than the *'LF'* and *'LF_NE'*, which is inline with the study conducted by Weber et al. (2019), hence the use of all feature (local and aggregated features) is more beneficial in terms of performance (all reported metrics) than making use of only local features. The scores obtained when reproducing the results for the RF classifier, using the suggested hyperparameters, were slightly higher than the ones reported in the original study (Weber et al., 2019) (shown in Table 3.2). The differences between the highest performance scores for the tested gradient boosting algorithms, against the best scores achieved by the benchmark, across all features sets were; 0% (Accuracy), *+3%* (Precision), *+1.1%* (Recall), and *+0.1%* (F1-Score). When directly comparing with the reported results from Weber et al. (2019) study, these values are slightly higher. From the evaluation conducted on account-level detection, each proposed gradient boosting algorithm outperformed our benchmark, with XGBoost being the most significant, having a difference of; *+1%* (Accuracy), *+0.6%* (Precision), *+2%* (Recall), and *+1.1%* (F1-Score). Additionally, comparing these results with those published in the original publication (Farrugia et al., 2020) that provided this dataset (shown in Table 3.1 in Section 3.1), we obtained better results in terms of F1-Score. Our proposed XGBoost, LGBM, and CatBoost obtained a difference of *+2.3%*, *+2%*, *+1.9%*, respectively, in terms of F1-Score, when compared to their best performing model (XGBoost with a different approach to tune hyperparameter than our proposed optimisation), which obtained a score of *0.960*. The positive difference in performance between our implementation of XGBoost and the one implemented by Farrugia et al. (2020), could be attributed to the number of hyperparameters tuned. According to Xia et al. (2017), having a vast search space to tune an XGBoost classifier with an efficient optimisation algorithm, can substantially improve performance and so, this study optimised ten hyperparameters in comparison to the three optimised by Farrugia et al. (2020). Results obtained from the supplementary dataset reinforced the notion that the proposed gradient boosting algorithms outperforms the benchmark RF, as the results obtained were similar to the account-level and transactional-level evaluations.

6.2 | Further improvements at Transactional-Level Detection using Data-Sampling Techniques

In Experiment 2 it was shown that data-sampling techniques did improve performance for the evaluated models, when applied on the *Elliptic dataset* (transactional-level detection). Notably, these techniques improved the overall Recall score, which inherently reduces the False Negative count. In particular, the XGBoost with default hyperparameters, in conjunction with NCL-SMOTE on the *'AF'* feature set, enhanced this score by 1%, from the previous highest score obtained (0.732) in Experiment 1. Similar improvement was achieved by the LGBM when this technique was applied, which further reinforces the assumption that making use of this technique achieves higher Recall (Junsomboon and Phienthrakul, 2017). In practice, this is vital as it determines how many illicit transactions go undetected, however, obtaining higher Recall sometimes can come at the expense of trading off a higher Precision score, as shown in our results (Table 5.10) and discussed by Bartoletti et al. (2018). Overall, an improvement in the other three metrics (Accuracy, Precision, F1-Score) could also be noted, with their statistical significance shown in Table 5.13. Similar results were obtained on the supplementary *NOAA dataset*, with the improvements being slightly better when compared to the findings obtained on the *Elliptic dataset*. It is worth noting that during these two evaluations (sampled

Elliptic and *NOAA*), the Recall score was higher when default hyperparameters were used. After tuning was applied, this score slightly degrades while the Precision score increases. This could be attributed to this method maximising the F1-Score while being oblivious to whether the score was attributed by good Precision or Recall, as outlined in the objective function set in the hyperparameter optimisation technique (Section 4.3.3).

6.3 | Adapting to Evolving Transactional Data-Streams

Shifting the focus to handle the problem of non-stationary data in the *Elliptic dataset*, in Experiment 3 it was shown that our adaptation of XGBoost to handle data-streams further improved the Recall score by 2.6%. Even though the overall F1-Score is slightly lower than the scores obtained in preceding experiments, Figures 5.10 and 5.11 show that all the evaluated adaptive learners seem to start recovering from the drop in performance (F1-Score) after timestep 43. It is worth noting that from all the F1-time plots (Figures 5.10 , 5.11, 5.12 and 5.13) in Experiment 3, our proposed adaptation seem to recover the fastest after the drop in performance (timestep 43). Moreover, the ARF learner obtained the highest scores in Accuracy, Precision and F1-Score during this experiment. Similarly to the reported results by Montiel et al. (2020), ARF outperformed the other evaluated models; however, our ASXGB achieved the highest scores in terms of Recall, and ranked second in all the other metrics. This shows that our adaptation of XGBoost is more efficient in terms of the reported metrics, when compared to the adaptation developed by Montiel et al. (2020), which was also evaluated during this test (transactional-level detection). Even though the adaptive learners obtained some slight improvements and adapted through the drop in performance when evaluated on transactional-level detection (after timestep 43), these learners performed relatively poorly when evaluated on the supplementary *NOAA* dataset, in a stream environment. When comparing the results with those obtained by Montiel et al. (2020), similar outcomes were obtained when they investigated both ARF and AXGB, in terms of Accuracy. They noted that this dataset had drifts with an unknown nature which is different, in comparison with the *Elliptic dataset* which contains sudden drifts.

7

Conclusion and Future Work

In this work, a solution to detect illicit activity on both an account and transactional level was presented. The study identified and tackled, inherent gaps in previous literature, including; (i) identifying which tree-based ensemble - Random Forest (RF) (Breiman, 2001) versus Gradient Boosting (Friedman, 2001), is more suitable in the identification of nefarious activity on the Blockchain, through a systematic evaluation including hypothesis testing (ii) comparing a wide range of heuristic-based data-sampling technique to counteract the effects of class imbalance (iii) shedding light on various techniques which aid to mitigate the effects of concept drift, which is often overlooked in this domain. Through an empirical evaluation, the potential application of decision tree-based gradient boosting algorithms, in conjunction with efficient hyperparameter optimisation and data-sampling techniques was presented. An adaptation of eXtreme Gradient Boosting (XGBoost) (Chen and Guestrin) to handle evolving data-streams (concept drift), with the utilisation of generalised stacking (Wolpert, 1992) to update the underlying ensemble (previously built learners which are no longer contributing to the overall prediction) was also proposed and investigated.

7.1 | Revisiting Aims and Objectives

In Chapter 1, we have defined several objectives. As a first objective the following was set; *Compare the selected state-of-the-art algorithms against each other, as well as against a chosen benchmark, to determine which model is most effective in the context of identifying licit-or-illicit activities on blockchain networks, at an account and transaction level.* To determine the most effective model in the context of identifying illicit activity at a transaction and account level, two datasets were selected; *Elliptic dataset* (Elliptic, 2020; Weber et al., 2019), which comprises of licit/illicit Bitcoin transactions (transactional-level detection) and the *Ethereum Illicit Accounts dataset* (Farrugia et al., 2020) which is made up of licit/illicit Ethereum accounts (account-level detection). This laid the foundation to conduct a systematic evaluation (including two statistical hypothesis tests, refer to Section 5.3) to compare and contrast various state-of-the-art gradient boosting algorithms, in particular; XGBoost (Chen and Guestrin), LGBM (Ke et al., 2017), and CatBoost (Prokhorenkova et al., 2018). Additionally, these were also compared against a selected benchmark, a RF classifier. From the results obtained in the first experiment, XGBoost and LGBM showed to be the most effective with respect to Precision, Recall and F1-Score. These best performing models obtained relatively similar results, with XGBoost performing better at an account-level, while LGBM performing better at a transactional-level, both of which were supported with hyperparameter optimisation. Moreover, when comparing these results against previous studies (Farrugia et al., 2020; Weber et al., 2019), performance was improved on both account and transactional level detection, with the most notable metric being F1-Score, where it was improved by 2.3% and 2.4%, respectively. It can be deduced that from the results outlined in Sections 5.3 and 6.1, this study has been able to fulfil hypothesis one (refer to Table 4.1) set out; therefore, it can be said that, yes, decision-tree based gradient boosting is more effective in terms of performance, when compared to RF in the context of this problem.

Moving forward to the second objective; *Improve the detection of licit-or-illicit activities for the selected state-of-the-art algorithms, through the adaptation of data sampling techniques, while also identifying which approach works best.* The utilisation of three approaches, NCL (Laurikkala, 2001), SMOTE (Chawla et al., 2002) and NCL-SMOTE (Junsomboon and Phienthrakul, 2017), were investigated to determine if Data-Sampling can further improve performance for the evaluated models. As discussed in Chapter 4, due to time limitations these approaches were only employed at a transactional-level. Overall, these techniques performed relatively the same and when compared (using statistical test) against the results obtained on the original dataset, the overall performance improved in terms of Precision, Recall and F1-Score. The most notable score was the Recall metric, which improved by 1% (using XGBoost in conjunction with NCL-SMOTE which is the best approach in this context). As noted in the first experiment, the False Negative count was quite significant when compared to the False Positive count. In practice, this is vital as it determines how many illicit transactions go undetected and so, reducing the False Negatives will capture more illicit transactions. It can be concluded that from the results outlined in Sections 5.4 and 6.2, this study has been able to accomplish hypothesis two (refer to Table 4.1) set out; therefore, it can be said that, yes, Data-Sampling techniques in conjunction with decision-tree based gradient boosting, can further improve performance.

Lastly, the third objective; *Improve the detection of licit-or-illicit activities at a transaction level, on state-of-the-art algorithms by handling concept drift more effectively in order to minimise performance degradation over time, thus enabling real-time transaction monitoring of cryptocurrency.* Several state-of-the-art adaptive learners were investigated to counteract concept drift, in the context of identifying illicit transactions on the Bitcoin network, in particular; Adaptive Random Forest (ARF) (Gomes et al., 2017), AXGB (Montiel et al., 2020) and our proposed approach ASXGB. From the conducted prequential-evaluation, it was evident that these adaptive learners are better than the models evaluated in the subsequent experiments, when handling shifting distributions over time. The F1-Score time plots have shown that our proposed adaptation, ASXGB, seems to recover relatively faster (after timestep 43, as shown in Figures 5.10, 5.11, 5.12 and 5.13) than the other tested adaptive learners. In addition, the proposed approach ranked second during this test, just behind ARF in terms of Precision and F1-Score; however, it was considered superior in terms of Recall, further increasing the score by 2.6% when compared against the results obtained from the second experiment. It can be deduced that from the results outlined in Sections 5.5 and 6.3, this study has been able to accomplish hypothesis three (refer to Table 4.1) set out; therefore, it can be said that, yes, Adaptive Stacked eXtreme Gradient Boosting (ASXGB) can be utilised to handle evolving data-streams, to reduce the effects of concept drift, thus showing potential for real-time transaction monitoring for cryptocurrency.

7.2 | Contributions

With the achievement of the objectives highlighted above, the main contributions of this work include;

- Demonstrating the effectiveness of Tree-structured Parzen Estimator (TPE) when applied in conjunction with XGBoost, LGBM and CatBoost. In our experiments we showed the increase performance of this approach when compared against the work proposed by Farrugia et al. (2020), using the *Ethereum Illicit Accounts dataset*, in terms of F1-Score.
- Further showing that decision tree-based gradient boosting algorithms, in particular Light Gradient Boosting Machine (LGBM), outperform Random Forest (RF) in the context of detecting illicit cryptocurrency transactions (improvement on the proposed solution by Weber et al. (2019)).

- Demonstrating that the data-sampling technique NCL-SMOTE, improves Recall across all the tested models when sampling the cryptocurrency transaction data, further reducing the False Negative Rate.
- Proposed an innovative adaptation of XGBoost, Adaptive Stacked eXtreme Gradient Boosting (ASXGB), that improves the handling of concept drift which is an important data distribution feature when dealing with transaction monitoring. From our experiments, our proposed ASXGB outperformed two state-of-the-art adaptive learners, namely ARF and AXGB in terms of Recall, when evaluated on the Elliptic transactional-level dataset and another concept drift benchmark dataset.

All the software developed leading to these contributions have been open-sourced and made publicly available on GitHub [1].

7.3 | Limitations

This proposed tool is far from perfect, and it has its limitations. It will only be able to sift through transactions or accounts as a means of detecting illicit activity, and so, follow-up manual work is also required to identify money laundering rings in this graph structure. The belief is that having an effective automated tool to detect this behaviour, coupled up with additional tools to visualise the graph structure (Singh and Best, 2019; Weber et al., 2019), may provide AML compliance with the explainability and support needed in the current climate of the cryptocurrency space. Another shortcoming is that the ground-truth in this domain is hard to validate, given that most of the labelling is done through clustering (Monamo et al., 2016), scraping online sources (i.e. forums) (Toyoda et al., 2017, 2018) or using heuristic based reasoning approaches (Weber et al., 2019); thus the results reported in this work and previous studies are based on subjective information. Moreover, the evaluation conducted on various adaptive learners, including our own proposed approach (ASXGB), did not utilise the power of Data-Sampling techniques nor did they undergo hyperparameter optimisation, even though prior experiments showed the effectiveness of these methods. This decision had to be taken as these procedures can add another layer of complexity when incorporated with adaptive learners training on an evolving data-stream, and given the time constraints we had to withdraw from investigating these techniques, leaving unanswered questions such as; "*Would hyperparameter tuning and Data-Sampling techniques further improve the performance for the evaluated adaptive learners ?*". Computational power was also not considered when

[1] AML-Crypto: `https://github.com/achmand/aml-crypto-graph`

evaluating our ASXGB approach, and given that the meta-learner keeps learning incrementally by appending new trees (as described in Section 4.3.4), this can become costly in terms of memory consumption. Lastly, it is worth noting that the evaluated models were tested on three datasets (one of them as supplementary data outside the domain) due to data being sensitive and difficult to find in this area. For this study the experiments were limited to the *Elliptic dataset* (transaction-level), *Ethereum Illicit Accounts dataset* (account-level) and the *NOAA weather dataset* (supplementary dataset).

7.4 | Future Work

There are multiple studies which can stem from this work; however, we propose the following prospects, which are based on the tasks we would have wanted to investigate if time was not a consideration;

- Investigate the use of how hyperparameter optimisation and data-sampling techniques, could have been integrated in our proposed adaptive learners ASXGB. One way to achieve this could have been to sample each training batch before it is fed to the underlying base learners and keep a reference of the previous N batches so as to tune newly created base learners before adding to the ensemble. This would add another layer of complexity/cost and should be investigated to find a balance between complexity and performance.

- As pointed out in Section 7.3, the proposed ASXGB does not handle the possibility of the meta learners' potential growth, which could result in memory issues. Techniques to handle such a problem could be investigated; for instance, once the meta-learner ensemble has grown to a specific threshold, one can prune older trees. Another approach could be to swap the XGBoost meta-learner with a simple perceptron. Monitor each individual base model using ADWIN (drift detector) (Bifet and Gavalda, 2007) and once a drift is detected, replace the base model with a new one, and reset the coefficients of the perceptron associated with that particular model, similarly to how *Ensemble of Restricted Hoeffding Trees* (Bifet et al., 2012) operate.

- Another prospect is to apply the same principles of ASXGB, to develop an adaptation of LGBM in order to handle evolving data-streams. It is known that LGBM can be trained in a much faster manner than XGBoost, given that it employs a technique called *'Gradient-Based One Side Sampling'* to downsample examples based on gradients (Ke et al., 2017; Sagi and Rokach, 2018). This proposal is worth pursuing since computation time is vital when employing models in a stream environment.

7.5 | Final Remarks

The tools proposed in this research were shown to work in a traditional-batch and stream environments, both of which can be valuable to stakeholders such as AML compliance departments and Law enforcement agencies. These tools can aid stakeholders by reducing the workload of manual inspections, which in turn can allow resources to be shifted elsewhere, such as, follow-up investigations from the suspected suspicious behaviour detected. Given the openness of Blockchain technology, the trail of payments can be traced, in order to pinpoint how these illicitly-gained funds are laundered into the financial system. Eventually, combating money laundering and other cryptocurrency-related crimes by the utilisation of detection systems, can reduce the risk for users and stakeholders of becoming victims of these crimes, which in turn encourages the adoption of Blockchain technology.

A

Ethereum Illicit Accounts Feature Set

The *Ethereum Illicit Accounts dataset* (Farrugia et al., 2020) contains a total of 42 features (extracted from the blockchain), with a total of 4,681 instances. All instances were labelled as illicit or licit accounts. The illicit accounts were acquired from Etherscamdb[1], whereby all illicit accounts listed as from the 17 of April 2019 were included in this dataset. The table below shows a complete list of the features extracted and included in this dataset.

[1] Etherscamdb: `http://etherscamdb.info/`

	Extracted Feature	Rank	Description	Data Type
1	**Avg_min_between_sent_tnx**	7	Average time between sent transactions for account in minutes	Integer
2	**Avg_min_between_received_tnx**	6	Average time between received transactions for account in minutes	Integer
3	**Time_Diff_between_first_and_last(Mins)**	1	Time difference between the first and last transaction	Integer
4	**Sent_tnx**	17	Total number of sent normal transactions	Integer
5	**Received_tnx**	18	Total number of received normal transactions	Integer
6	**Number_of_Created_Contracts**	28	Total Number of created contract transactions	Integer
7	**Unique_Received_From_Addresses**	8	Total Unique addresses from which account received transactions	Integer
8	**Unique_Sent_To_Addresses**	20	Total Unique addresses from which account sent transactions	Integer
9	**Min_Value_Received**	3	Minimum value in Ether ever received	Double
10	**Max_Value_Received**	9	Maximum value in Ether ever received	Double
11	**Avg_Value_Received**	5	Average value in Ether ever received	Double
12	**Min_Val_Sent**	4	Minimum value of Ether ever sent	Double
13	**Max_Val_Sent**	13	Maximum value of Ether ever sent	Double
14	**Avg_Val_Sent**	10	Average value of Ether ever sent	Double
15	**Min_Value_Sent_To_Contract**	39	Minimum value of Ether sent to a contract	Double
16	**Max_Value_Sent_To_Contract**	40	Maximum value of Ether sent to a contract	Double
17	**Avg_Value_Sent_To_Contract**	41	Average value of Ether sent to contracts	Double
18	**Total_Transactions(Including_Tnx_to_Create_Contract)**	12	Total number of transactions	Integer
19	**Total_Ether_Sent**	14	Total Ether sent for account address	Double
20	**Total_Ether_Received**	11	Total Ether received for account address	Double
21	**Total_Ether_Sent_Contracts**	38	Total Ether sent to Contract addresses	Double
22	**Total_Ether_Balance**	2	Total Ether Balance following enacted transactions	Double
23	**Total_ERC20_Tnxs**	16	Total number of ERC20 token transfer transactions	Integer
24	**ERC20_Total_Ether_Received**	19	Total ERC20 token received transactions in Ether	Double
25	**ERC20_Total_Ether_Sent**	24	Total ERC20 token sent transactions in Ether	Double
26	**ERC20_Total_Ether_Sent_Contract**	37	Total ERC20 token transfer to other contracts in Ether	Double
27	**ERC20_Uniq_Sent_Addr**	26	Number of ERC20 token transactions sent to Unique account addresses	Integer
28	**ERC20_Uniq_Rec_Addr**	25	Number of ERC20 token transactions received from Unique addresses	Interger
29	**ERC20_Uniq_Rec_Contract_Addr**	23	Number of ERC20 token transactions received from Unique contract addresses	Integer
30	**ERC20_Avg_Time_Between_Sent_Tnx**	31	Average time between ERC20 token sent transactions in minutes	Integer
31	**ERC20_Avg_Time_Between_Rec_Tnx**	33	Average time between ERC20 token received transactions in minutes	Integer
32	**ERC20_Avg_Time_Between_Contract_Tnx**	32	Average time ERC20 token between sent token transactions	Integer
33	**ERC20_Min_Val_Rec**	15	Minimum value in Ether received from ERC20 token transactions for account	Double
34	**ERC20_Max_Val_Rec**	21	Maximum value in Ether received from ERC20 token transactions for account	Double
35	**ERC20_Avg_Val_Rec**	22	Average value in Ether received from ERC20 token transactions for account	Double
36	**ERC20_Min_Val_Sent**	27	Minimum value in Ether sent from ERC20 token transactions for account	Double
37	**ERC20_Max_Val_Sent**	29	Maximum value in Ether sent from ERC20 token transactions for account	Double
38	**ERC20_Avg_Val_Sent**	30	Average value in Ether sent from ERC20 token transactions for account	Double
39	**ERC20_Uniq_Sent_Token_Name**	34	Number of Unique ERC20 tokens transferred	Integer
40	**ERC20_Uniq_Rec_Token_Name**	36	Number of Unique ERC20 tokens received	Integer
41	**ERC20_Most_Sent_Token_Type**	42	Most sent token for account via ERC20 transaction	String
42	**ERC20_Most_Rec_Token_Type**	35	Most received token for account via ERC20 transaction	String

B

Optimal Hyperparameters from TPE

B.1 | Experiment 1

```
{'n_estimators': 1585, 'learning_rate': 0.050654853295849546, 'max_depth': 5, '
    ↪ subsample': 0.9317812643350265, 'colsample_bytree': 0.6367700203690464, '
    ↪ colsample_bylevel': 0.9168972013604229, 'min_child_weight':
    ↪ 1.1914420820435448e-06, 'reg_alpha': 1.1044975415552232e-06, 'reg_lambda
    ↪ ': 2.3419129189402086e-05, 'gamma': 0.000878097449991818}
```

Listing B.1: Optimal Hyperparameters for XGBoost - Elliptic Dataset 'LF' Feature set

```
{'n_estimators': 4885, 'learning_rate': 0.04339929288335128, 'max_depth': 3, '
    ↪ subsample': 0.743276507002905, 'colsample_bytree': 0.7960358287306276, '
    ↪ colsample_bylevel': 0.9179800965039797, 'min_child_weight':
    ↪ 0.0010558271063347655, 'reg_alpha': 0.00020086090356417362, 'reg_lambda':
    ↪  7.11326230069552e-07, 'gamma': 1.2366784965997972e-06}
```

Listing B.2: Optimal Hyperparameters for XGBoost - Elliptic Dataset 'LF_NE' Feature set

```
{'n_estimators': 475, 'learning_rate': 0.17180405341332797, 'max_depth': 5, '
    ↪ subsample': 0.7107204492692905, 'colsample_bytree': 0.7610132807939545, '
    ↪ colsample_bylevel': 0.6399663301477401, 'min_child_weight':
    ↪ 0.021463723370857116, 'reg_alpha': 0.00013176587726054908, 'reg_lambda':
    ↪ 5.3824033197120364e-06, 'gamma': 1.4897607985369035e-05}
```

Listing B.3: Optimal Hyperparameters for XGBoost - Elliptic Dataset 'AF' Feature set

```
{n_estimators': 2561, 'learning_rate': 0.07425551644920138, 'max_depth': 3, '
    ↪ subsample': 0.8456188891447396, 'colsample_bytree': 0.8625494671828154, '
    ↪ colsample_bylevel': 0.662579769175881, 'min_child_weight':
    ↪ 5.725233439812487e-05, 'reg_alpha': 3.440356098019968e-06, 'reg_lambda':
    ↪ 0.009046739391051883, 'gamma': 0.0002974922404499394}
```

Listing B.4: Optimal Hyperparameters for XGBoost - Elliptic Dataset 'AF_NE' Feature set

```
{'n_estimators': 3888, 'subsample_freq': 1, 'learning_rate':
    ↪ 0.005284256053565403, 'num_leaves': 56, 'colsample_bytree':
    ↪ 0.5867440901122031, 'subsample': 0.9375299749093455, 'min_child_samples':
    ↪  66, 'min_child_weight': 0.014264760992663937, 'reg_alpha':
    ↪ 0.00021668874539322757, 'reg_lambda': 2.997329007143576e-05}
```

Listing B.5: Optimal Hyperparameters for LGBM - Elliptic Dataset 'LF' Feature set

```
{'n_estimators': 2233, 'subsample_freq': 1, 'learning_rate':
    ↪ 0.02599555713447133, 'num_leaves': 802, 'colsample_bytree':
    ↪ 0.7538417724596406, 'subsample': 0.8973858606222604, 'min_child_samples':
    ↪  394, 'min_child_weight': 0.0006913272092669252, 'reg_alpha':
    ↪ 4.5653205753082604e-07, 'reg_lambda': 1.7133163555853147e-05}
```

Listing B.6: Optimal Hyperparameters for LGBM - Elliptic Dataset 'LF_NE' Feature set

```
{'n_estimators': 1036, 'subsample_freq': 1, 'learning_rate':
    ↪ 0.04512826974883081, 'num_leaves': 359, 'colsample_bytree':
    ↪ 0.8806545920095481, 'subsample': 0.9630402764474666, 'min_child_samples':
    ↪  327, 'min_child_weight': 4.3230516180285375e-05, 'reg_alpha':
    ↪ 1.1165994854235954e-05, 'reg_lambda': 2.4195307322425423e-05}
```

Listing B.7: Optimal Hyperparameters for LGBM - Elliptic Dataset 'AF' Feature set

```
{'n_estimators': 4261, 'subsample_freq': 1, 'learning_rate':
    ↪ 0.09722424033543176, 'num_leaves': 770, 'colsample_bytree':
    ↪ 0.6478101618090795, 'subsample': 0.6139853128533745, 'min_child_samples':
    ↪  255, 'min_child_weight': 8.479757535852648e-06, 'reg_alpha':
    ↪ 1.3090333096292452e-07, 'reg_lambda': 0.005951314850246193}
```

Listing B.8: Optimal Hyperparameters for LGBM - Elliptic Dataset 'AF_NE' Feature set

```
{'iterations': 3407, 'bootstrap_type': 'MVS', 'learning_rate':
    ↪ 0.09729275852441788, 'depth': 8, 'random_strength': 18, 'l2_leaf_reg':
    ↪ 3.609123468399048, 'subsample': 0.9836283326148988, '
    ↪ leaf_estimation_iterations': 7, 'rsm': 0.3774999976158142}
```

Listing B.9: Optimal Hyperparameters for CatBoost - Elliptic Dataset 'LF' Feature set

```
{'iterations': 3766, 'bootstrap_type': 'MVS', 'learning_rate':
    ↪ 0.027700575068593025, 'depth': 4, 'random_strength': 20, 'l2_leaf_reg':
    ↪ 1.000316858291626, 'subsample': 0.5841074585914612, '
    ↪ leaf_estimation_iterations': 8, 'rsm': 0.4650000035762787}
```

Listing B.10: Optimal Hyperparameters for CatBoost - Elliptic Dataset 'LF_NE' Feature set

```
{'iterations': 3915, 'bootstrap_type': 'MVS', 'learning_rate':
    ↪ 0.03919879347085953, 'depth': 5, 'random_strength': 12, 'l2_leaf_reg':
    ↪ 1.466575026512146, 'subsample': 0.6056259274482727, '
    ↪ leaf_estimation_iterations': 7, 'rsm': 0.8224999904632568}
```

Listing B.11: Optimal Hyperparameters for CatBoost - Elliptic Dataset 'AF' Feature set

```
{'iterations': 2203, 'bootstrap_type': 'MVS', 'learning_rate':
    ↪ 0.12073484808206558, 'depth': 4, 'random_strength': 8, 'l2_leaf_reg':
    ↪ 1.1098747253417969, 'subsample': 0.6365466117858887, '
    ↪ leaf_estimation_iterations': 6, 'rsm': 0.8700000047683716}
```

Listing B.12: Optimal Hyperparameters for CatBoost - Elliptic Dataset 'AF_NE' Feature set

```
{'n_estimators': 431, 'learning_rate': 0.07853386099915022, 'max_depth': 3, '
    ↪ subsample': 0.6788149328377757, 'colsample_bytree': 0.9055926731831192, '
    ↪ colsample_bylevel': 0.8150557185716437, 'min_child_weight':
    ↪ 6.543265212724884e-05, 'reg_alpha': 1.3075090623870832e-07, 'reg_lambda':
    ↪  0.0012272847930438772, 'gamma': 0.035718743636794725}
```

Listing B.13: Optimal Hyperparameters for XGBoost - Ethereum Illicit Accounts Dataset

```
{'n_estimators': 122, 'subsample_freq': 1, 'learning_rate': 0.19232892325747805,
    ↪  'num_leaves': 693, 'colsample_bytree': 0.7779646637843973, 'subsample':
    ↪ 0.7061136386427012, 'min_child_weight': 0.0001519376996130637, 'reg_alpha
    ↪ ': 0.0007499216305329894, 'reg_lambda': 2.3537581461084842e-05}
```

Listing B.14: Optimal Hyperparameters for LGBM - Ethereum Illicit Accounts Dataset

```
{'iterations': 130, 'bootstrap_type': 'MVS', 'learning_rate': 0.1196994632, '
    ↪ depth': 5, 'random_strength': 2, 'l2_leaf_reg': 2.028006554, 'subsample':
    ↪  0.788125813, 'leaf_estimation_iterations': 6, 'rsm': 0.2849999964}
```

Listing B.15: Optimal Hyperparameters for CatBoost - Ethereum Illicit Accounts Dataset

```
{'n_estimators': 750, 'max_samples': 0.9989555808064626, 'max_features': 'sqrt'}
```

Listing B.16: Optimal Hyperparameters for RF - Ethereum Illicit Accounts Dataset

```
{'n_estimators': 686, 'learning_rate': 0.10278359593536876, 'max_depth': 3, '
    ↪ subsample': 0.9669853429860304, 'colsample_bytree': 0.9390481435192746, '
    ↪ colsample_bylevel': 0.8259357777868539, 'min_child_weight':
    ↪ 0.23006847451636644, 'reg_alpha': 0.9698309151946589, 'reg_lambda':
    ↪ 3.6482620406500425e-05, 'gamma': 9.856689719822213e-05}
```

Listing B.17: Optimal Hyperparameters for XGBoost - NOAA Dataset

```
{'n_estimators': 970, 'subsample_freq': 1, 'learning_rate': 0.00942139530205555,
    ↪  'num_leaves': 247, 'colsample_bytree': 0.9042782299701083, 'subsample':
    ↪ 0.6496983898551796, 'min_child_weight': 0.2509419055407807, 'reg_alpha':
    ↪ 0.07504697557719497, 'reg_lambda': 0.007102258113567796}
```

Listing B.18: Optimal Hyperparameters for LGBM - NOAA Dataset

```
{'iterations': 2201, 'bootstrap_type': 'MVS', 'learning_rate': 0.0295537468, '
    ↪ depth': 9, 'random_strength': 18, 'l2_leaf_reg': 5.976006031, 'subsample
    ↪ ': 0.5883533955, 'leaf_estimation_iterations': 4, 'rsm': 0.2849999964}
```

Listing B.19: Optimal Hyperparameters for CatBoost - NOAA Dataset

```
{'n_estimators': 950, 'max_samples': 0.7803639907513293, 'max_features': 'sqrt'}
```

Listing B.20: Optimal Hyperparameters for RF - NOAA Dataset

B.2 | Experiment 2

```
{'n_estimators': 4488, 'learning_rate': 0.043129412229658555, 'max_depth': 4, '
    ↪ subsample': 0.8752930405492565, 'colsample_bytree': 0.5022213841856625, '
    ↪ colsample_bylevel': 0.7294422822973423, 'min_child_weight':
    ↪ 0.0002551276759986131, 'reg_alpha': 0.00019202859809111611, 'reg_lambda':
    ↪  0.024104669001685553, 'gamma': 3.42181202265282e-05}
```

Listing B.21: Optimal Hyperparameters for XGBoost - Elliptic Dataset 'AF' Feature set when sampled with Neighbourhood Cleaning Rule (NCL).

```
{'n_estimators': 1260, 'learning_rate': 0.08603443933689565, 'max_depth': 3, '
    ↪ subsample': 0.8832485951402574, 'colsample_bytree': 0.5592158861348477, '
    ↪ colsample_bylevel': 0.7251182439618324, 'min_child_weight':
```

```
    ↪ 0.05385358473992219, 'reg_alpha': 5.5392592787449286e-06, 'reg_lambda':
    ↪ 1.973024595870968e-05, 'gamma': 0.0006031140152965155}
```

Listing B.22: Optimal Hyperparameters for XGBoost - Elliptic Dataset 'AF_NE' Feature set when sampled with Neighbourhood Cleaning Rule (NCL).

```
{'n_estimators': 2935, 'subsample_freq': 1, 'learning_rate': 0.0423035443405413,
    ↪  'num_leaves': 821, 'colsample_bytree': 0.729082769944796, 'subsample':
    ↪ 0.9558766592739659, 'min_child_samples': 303, 'min_child_weight':
    ↪ 0.07736806864321738, 'reg_alpha': 0.0006461317856670983, 'reg_lambda':
    ↪ 3.2252712450565137e-06}
```

Listing B.23: Optimal Hyperparameters for LGBM - Elliptic Dataset 'AF' Feature set when sampled with Neighbourhood Cleaning Rule (NCL).

```
{'n_estimators': 473, 'subsample_freq': 1, 'learning_rate': 0.14489639312813868,
    ↪  'num_leaves': 15, 'colsample_bytree': 0.9501052853091929, 'subsample':
    ↪ 0.9999922162097159, 'min_child_samples': 398, 'min_child_weight':
    ↪ 0.000572827524130768, 'reg_alpha': 0.00015695279114051537, 'reg_lambda':
    ↪ 1.1835883344548447e-07}
```

Listing B.24: Optimal Hyperparameters for LGBM - Elliptic Dataset 'AF_NE' Feature set when sampled with Neighbourhood Cleaning Rule (NCL).

```
{'n_estimators': 300, 'max_samples': 0.98203901258471, 'max_features': 'sqrt'}
```

Listing B.25: Optimal Hyperparameters for RF - Elliptic Dataset 'AF' Feature set when sampled with Neighbourhood Cleaning Rule (NCL).

```
{'n_estimators': 550, 'max_samples': 0.9152422886808692, 'max_features': 'sqrt'}
```

Listing B.26: Optimal Hyperparameters for RF - Elliptic Dataset 'AF_NE' Feature set when sampled with Neighbourhood Cleaning Rule (NCL).

```
{'n_estimators': 2468, 'learning_rate': 0.016555234134056844, 'max_depth': 7, '
    ↪ subsample':0.6772966101967116, 'colsample_bytree': 0.8750922554061572, '
    ↪ colsample_bylevel': 0.541908774849356, 'min_child_weight':
    ↪ 5.2634493135976105e-06, 'reg_alpha': 0.00013782852828065698, 'reg_lambda
    ↪ ': 1.9144408763587467e-05, 'gamma': 1.2179673308658027e-06}
```

Listing B.27: Optimal Hyperparameters for XGBoost - Elliptic Dataset 'AF' Feature set when sampled with Synthetic Minority Over-Sampling (SMOTE).

```
{'n_estimators': 4345, 'learning_rate': 0.015002789614188404, 'max_depth': 5, '
    ↪ subsample': 0.6864374504940947, 'colsample_bytree': 0.8227730263394168, '
    ↪ colsample_bylevel': 0.564203782287485, 'min_child_weight':
    ↪ 2.7821335143885406e-06, 'reg_alpha': 0.0003838867027467089, 'reg_lambda':
    ↪  1.4126445257677391e-05, 'gamma': 2.825657372266011e-07}
```

Listing B.28: Optimal Hyperparameters for XGBoost - Elliptic Dataset 'AF_NE' Feature set when sampled with Synthetic Minority Over-Sampling (SMOTE).

```
{'n_estimators': 1518, 'subsample_freq': 1, 'learning_rate':
    ↪ 0.02686341441883127, 'num_leaves': 693, 'colsample_bytree':
    ↪ 0.5575750904771983, 'subsample': 0.9312752561546436, 'min_child_samples':
    ↪  392, 'min_child_weight': 0.0002417019088245615, 'reg_alpha':
    ↪ 3.0210017424703597e-05, 'reg_lambda': 1.9336471177832827e-05}
```

Listing B.29: Optimal Hyperparameters for LGBM - Elliptic Dataset 'AF' Feature set when sampled with Synthetic Minority Over-Sampling (SMOTE).

```
{'n_estimators': 4039, 'subsample_freq': 1, 'learning_rate':
    ↪ 0.016592784481232123, 'num_leaves': 533, 'colsample_bytree':
    ↪ 0.6148311617055384, 'subsample': 0.5486704069744622, 'min_child_samples':
    ↪  354, 'min_child_weight': 2.536410266625732e-06, 'reg_alpha':
    ↪ 2.6886397300480966e-05, 'reg_lambda': 1.1511536602335924e-07}
```

Listing B.30: Optimal Hyperparameters for LGBM - Elliptic Dataset 'AF_NE' Feature set when sampled with Synthetic Minority Over-Sampling (SMOTE).

```
{'n_estimators': 900, 'max_samples': 0.9924762782693718, 'max_features': 'sqrt'}
```

Listing B.31: Optimal Hyperparameters for RF - Elliptic Dataset 'AF' Feature set when sampled with Synthetic Minority Over-Sampling (SMOTE).

```
{'n_estimators': 600, 'max_samples': 0.9718394105468453, 'max_features': 'log2'}
```

Listing B.32: Optimal Hyperparameters for RF - Elliptic Dataset 'AF_NE' Feature set when sampled with Synthetic Minority Over-Sampling (SMOTE).

```
{'n_estimators': 1149, 'learning_rate': 0.03451669333808158, 'max_depth': 7, '
    ↪ subsample':0.5635583977428819, 'colsample_bytree': 0.6366853270101964, '
    ↪ colsample_bylevel': 0.7893054408663991, 'min_child_weight':
    ↪ 4.739987190030236e-06, 'reg_alpha': 4.6693396003499663e-07, 'reg_lambda':
    ↪  0.0003865526336625385, 'gamma': 2.6081434499725793e-05}
```

Listing B.33: Optimal Hyperparameters for XGBoost - Elliptic Dataset 'AF' Feature set when sampled with NCL-SMOTE.

```
{'n_estimators': 948, 'learning_rate': 0.1026141189677146, 'max_depth': 7, '
    ↪ subsample': 0.5468285060264697, 'colsample_bytree': 0.675624306763575, '
    ↪ colsample_bylevel': 0.7943840681950523, 'min_child_weight':
    ↪ 0.00026581370473607944, 'reg_alpha': 1.4494911525904446e-06, 'reg_lambda
    ↪ ': 4.770670121334852e-07, 'gamma': 3.3282488912901886e-06}
```

Listing B.34: Optimal Hyperparameters for XGBoost - Elliptic Dataset 'AF_NE' Feature set when sampled with NCL-SMOTE.

```
{'n_estimators': 4178, 'subsample_freq': 1, 'learning_rate':
    ↪ 0.008514035475086324, 'num_leaves': 71, 'colsample_bytree':
    ↪ 0.9886521546582805, 'subsample': 0.5832400330945102, 'min_child_samples':
    ↪  330, 'min_child_weight': 0.000256385409520096, 'reg_alpha':
    ↪ 2.7367542903521125e-07, 'reg_lambda': 0.0001199777277505261}
```

Listing B.35: Optimal Hyperparameters for LGBM - Elliptic Dataset 'AF' Feature set when sampled with NCL-SMOTE.

```
{'n_estimators': 834, 'subsample_freq': 1, 'learning_rate':
    ↪ 0.043944880208222466, 'num_leaves': 326, 'colsample_bytree':
    ↪ 0.5726058643516077, 'subsample': 0.810451897504235, 'min_child_samples':
    ↪ 284, 'min_child_weight': 5.832137501040796e-06, 'reg_alpha':
    ↪ 5.762231119059793e-05, 'reg_lambda': 7.152316529235852e-05}
```

Listing B.36: Optimal Hyperparameters for LGBM - Elliptic Dataset 'AF_NE' Feature set when sampled with NCL-SMOTE.

```
{'n_estimators': 950, 'max_samples': 0.9971050029293645, 'max_features': 'sqrt'}
```

Listing B.37: Optimal Hyperparameters for RF - Elliptic Dataset 'AF' Feature set when sampled with NCL-SMOTE.

```
{'n_estimators': 850, 'max_samples': 0.9999638105412046, 'max_features': 'log2'}
```

Listing B.38: Optimal Hyperparameters for RF - Elliptic Dataset 'AF_NE' Feature set when sampled with NCL-SMOTE.

```
{'n_estimators': 2638, 'learning_rate': 0.050927485269579784, 'max_depth': 9, '
    ↪ subsample': 0.8850022065964945, 'colsample_bytree': 0.8850022065964945, '
    ↪ colsample_bylevel': 0.8599759362813014, 'min_child_weight':
    ↪ 0.009673382756603283, 'reg_alpha': 9.539856882337921e-05, 'reg_lambda':
    ↪ 1.1330084820491268e-05, 'gamma': 4.2937561466681195e-06}
```

Listing B.39: Optimal Hyperparameters for XGBoost - NOAA Dataset when sampled with NCL.

```
{'n_estimators': 2078, 'subsample_freq': 1, 'learning_rate':
    ↪ 0.013352049256900976, 'num_leaves': 549, 'colsample_bytree':
    ↪ 0.7105710021662228, 'subsample': 0.8109380992594621, 'min_child_samples':
    ↪  54, 'min_child_weight': 0.008626487671101805, 'reg_alpha':
    ↪ 5.632256703697636e-06, 'reg_lambda': 0.01828152288737052}
```

Listing B.40: Optimal Hyperparameters for LGBM - NOAA Dataset when sampled with NCL.

```
{'n_estimators': 800, 'max_samples': 0.9915293070816315, 'max_features': 'log2'}
```

Listing B.41: Optimal Hyperparameters for RF - NOAA Dataset when sampled with NCL.

```
{'n_estimators': 4664, 'learning_rate': 0.03896337896870269, 'max_depth': 10, '
    ↪ subsample': 0.671870137970136, 'colsample_bytree': 0.9023492619794203, '
    ↪ colsample_bylevel':0.7821187110195753, 'min_child_weight':
    ↪ 0.0009120440224687303, 'reg_alpha': 0.0004965315185823202, 'reg_lambda':
    ↪ 0.006362939210948986, 'gamma': 0.002531642946263811}
```

Listing B.42: Optimal Hyperparameters for XGBoost - NOAA Dataset when sampled with SMOTE.

```
{'n_estimators': 3838, 'subsample_freq': 1, 'learning_rate':
    ↪ 0.018885526001721917, 'num_leaves': 531, 'colsample_bytree':
    ↪ 0.9798987810659789, 'subsample': 0.6330725162984325, 'min_child_samples':
    ↪  1, 'min_child_weight': 0.000244933461981523, 'reg_alpha':
    ↪ 0.012506700853197978, 'reg_lambda': 1.750875985194234e-06}
```

Listing B.43: Optimal Hyperparameters for LGBM - NOAA Dataset when sampled with SMOTE.

```
{'n_estimators': 750, 'max_samples': 0.9564372131684578, 'max_features': 'log2'}
```

Listing B.44: Optimal Hyperparameters for RF - NOAA Dataset when sampled with SMOTE.

```
{'n_estimators': 4596, 'learning_rate': 0.04038196423627139, 'max_depth': 6, '
    ↪ subsample': 0.7154191406875013, 'colsample_bytree': 0.7267352433483205, '
    ↪ colsample_bylevel': 0.8826277282486175, 'min_child_weight':
    ↪ 0.0005482950190124504, 'reg_alpha': 5.676983451968034e-06, 'reg_lambda':
    ↪ 7.23325209637167e-05, 'gamma': 0.0007014525132025484}
```

Listing B.45: Optimal Hyperparameters for XGBoost - NOAA Dataset when sampled with NCL-SMOTE.

```
{'n_estimators': 1095, 'subsample_freq': 1, 'learning_rate': 0.0493344260557109,
    ↪  'num_leaves': 175, 'colsample_bytree': 0.8494418719270661, 'subsample':
    ↪ 0.9398266126294899, 'min_child_samples': 25, 'min_child_weight':
    ↪ 1.3492254178729675e-05, 'reg_alpha': 0.002434526892861787, 'reg_lambda':
    ↪ 1.3404496793037812e-07}
```

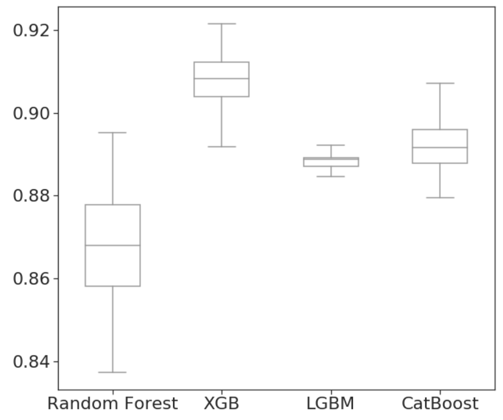
Listing B.46: Optimal Hyperparameters for LGBM - NOAA Dataset when sampled with NCL-SMOTE.

```
{'n_estimators': 150, 'max_samples': 0.8132814756379779, 'max_features': 'sqrt'}
```

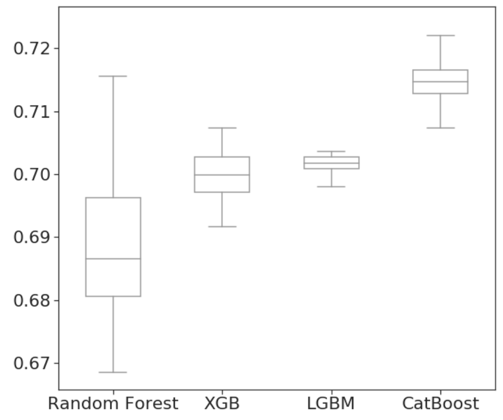
Listing B.47: Optimal Hyperparameters for RF - NOAA Dataset when sampled with NCL-SMOTE.

C

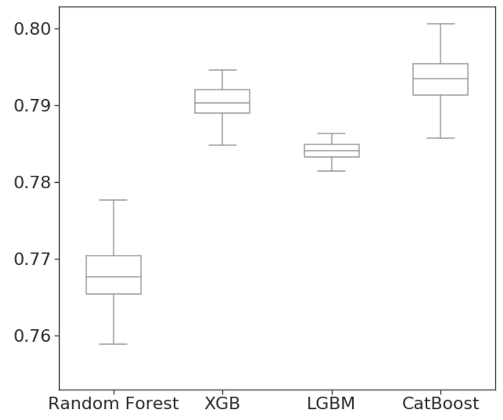
Supplementary Results



(a) Precision

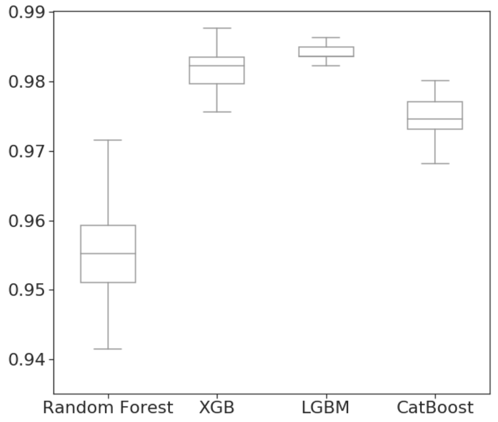


(b) Recall

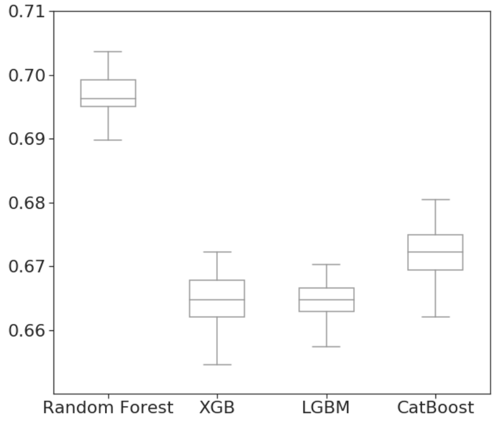


(c) F1-Score

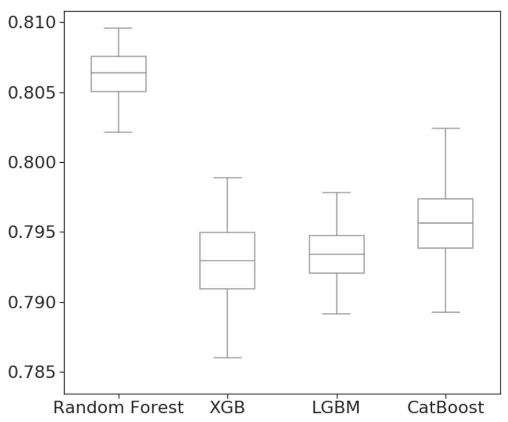
Figure C.1: Box-plots for Precision, Recall and F1-Score, when evaluating tuned models and the benchmark with suggested hyperparameters, on the *Elliptic dataset* using the '*LF*' feature set (Experiment 1: Transactional-Level Detection).



(a) Precision

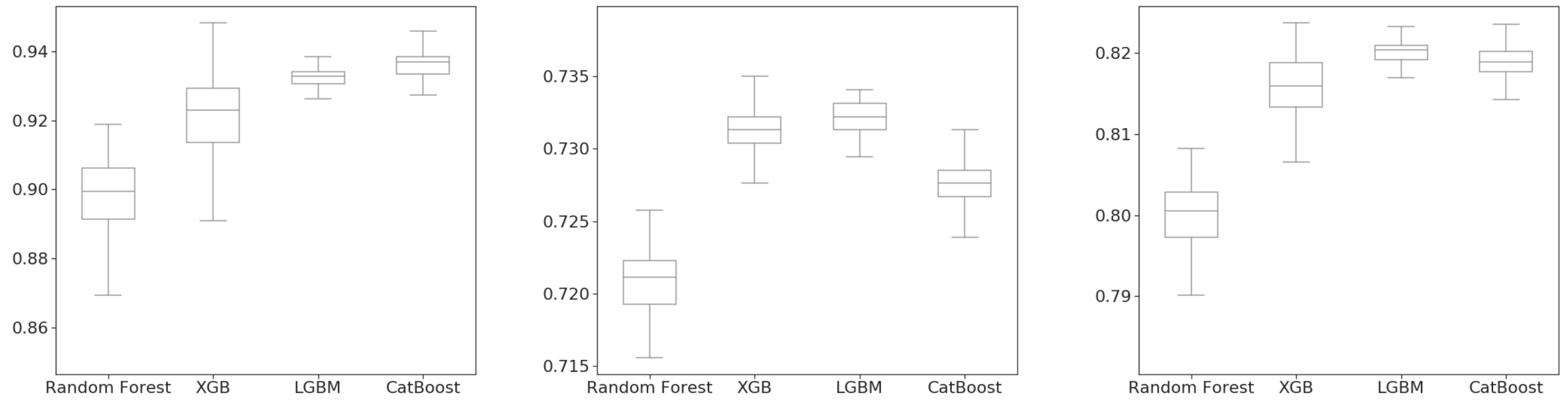


(b) Recall



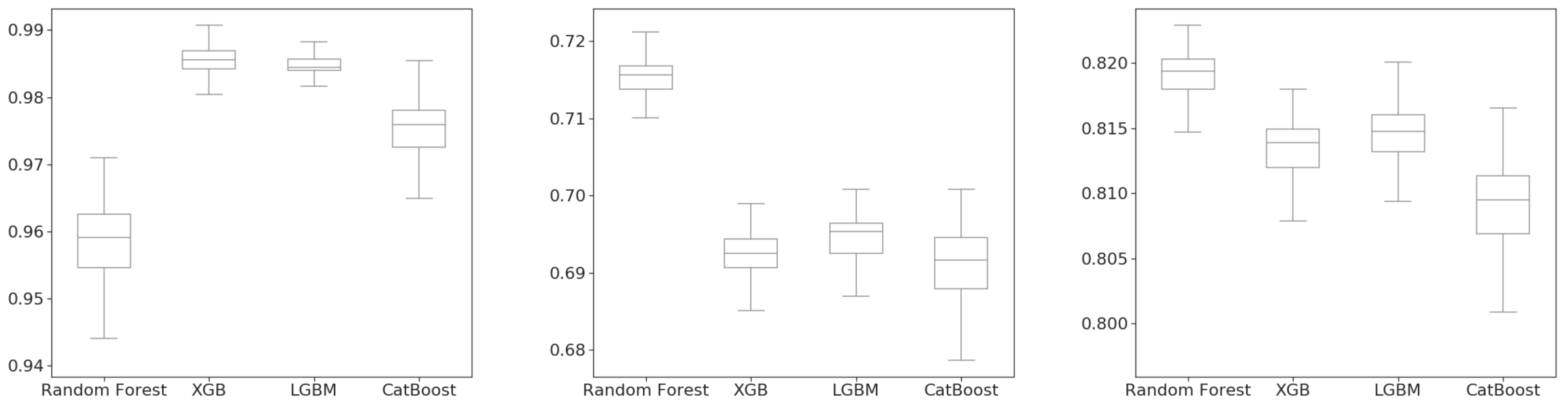
(c) F1-Score

Figure C.2: Box-plots for Precision, Recall and F1-Score, when evaluating tuned models and the benchmark with suggested hyperparameters, on the *Elliptic dataset* using the '*LF_NE*' feature set (Experiment 1: Transactional-Level Detection).



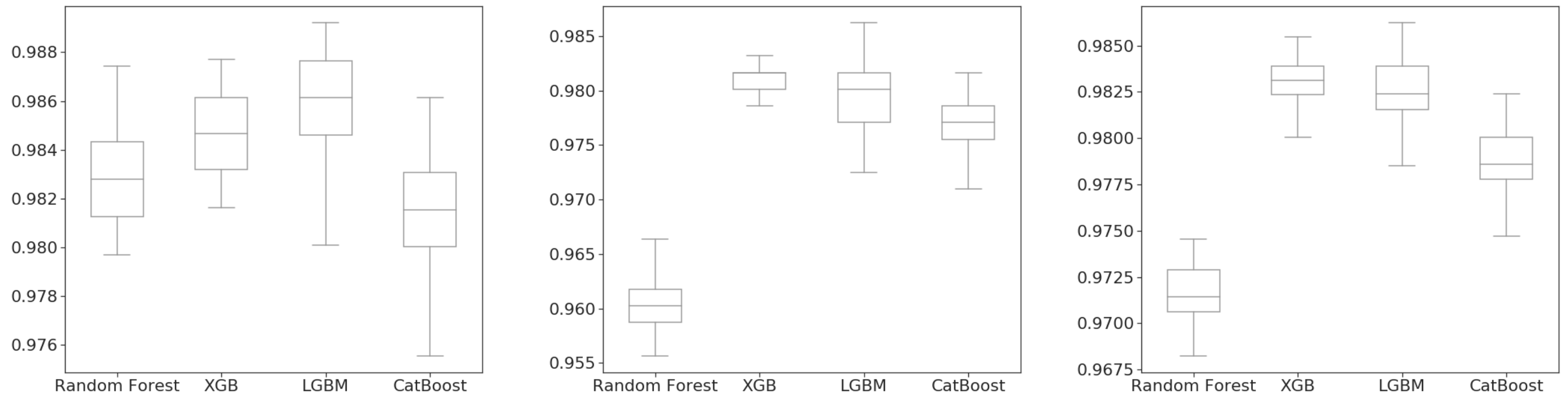
(a) Precision (b) Recall (c) F1-Score

Figure C.3: Box-plots for Precision, Recall and F1-Score, when evaluating tuned models and the benchmark with suggested hyperparameters, on the *Elliptic dataset* using the '*AF*' feature set (Experiment 1: Transactional-Level Detection).



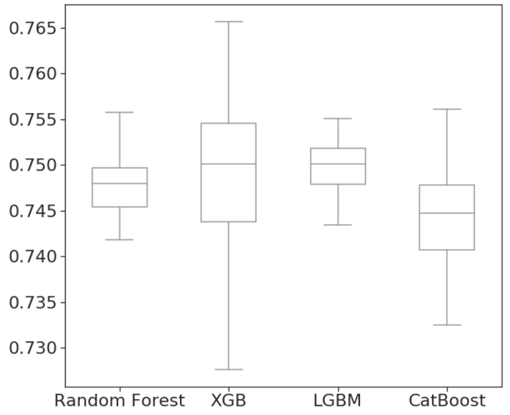
(a) Precision (b) Recall (c) F1-Score

Figure C.4: Box-plots for Precision, Recall and F1-Score, when evaluating tuned models and the benchmark with suggested hyperparameters, on the *Elliptic dataset* using the '*AF_NE*' feature set (Experiment 1: Transactional-Level Detection).

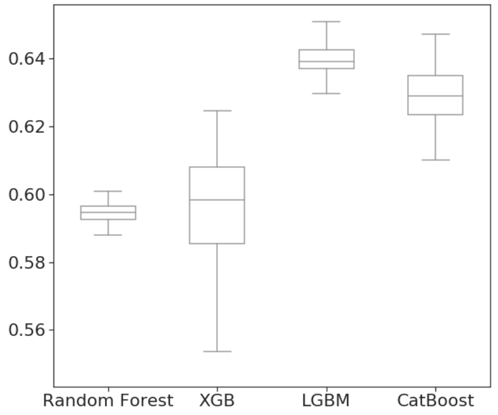


(a) Precision (b) Recall (c) F1-Score

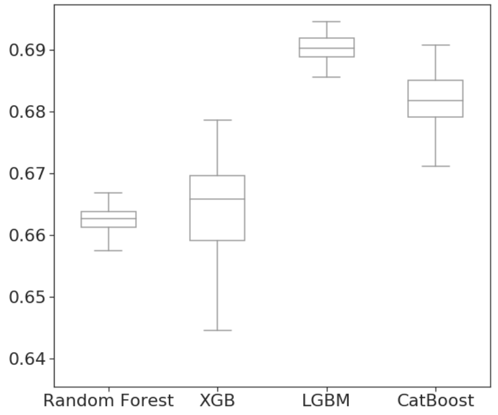
Figure C.5: Box-plots for Precision, Recall and F1-Score, when evaluating tuned models on the *Ethereum Illicit Accounts dataset* (Experiment 1: Account-Level Detection).



(a) Precision

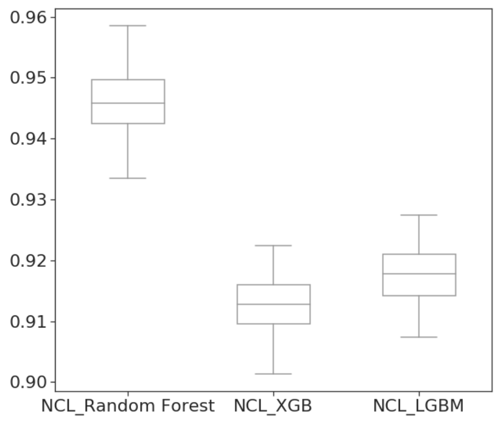


(b) Recall

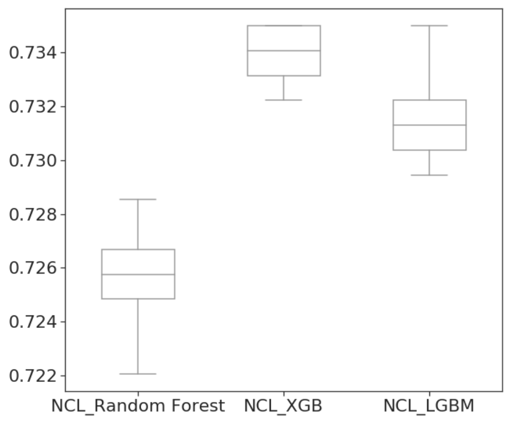


(c) F1-Score

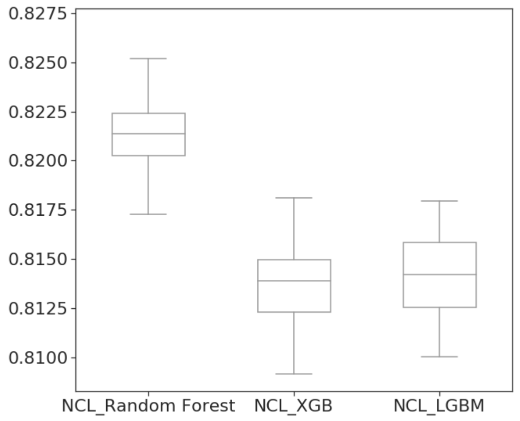
Figure C.6: Box-plots for Precision, Recall and F1-Score, when evaluating tuned models on the *NOAA dataset* (Experiment 1: Supplementary Dataset).



(a) Precision

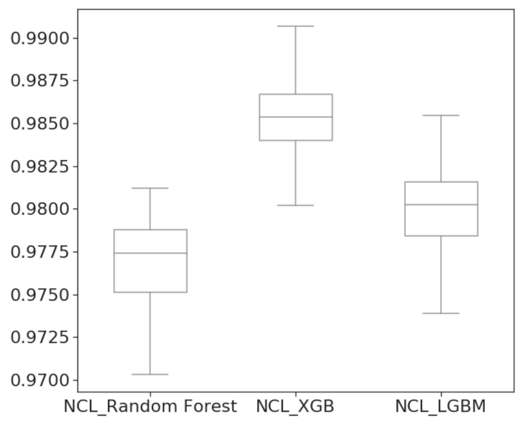


(b) Recall

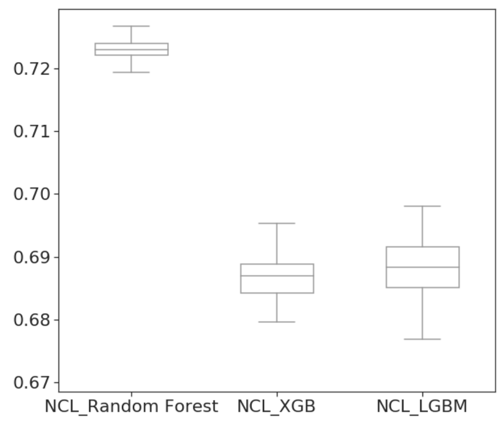


(c) F1-Score

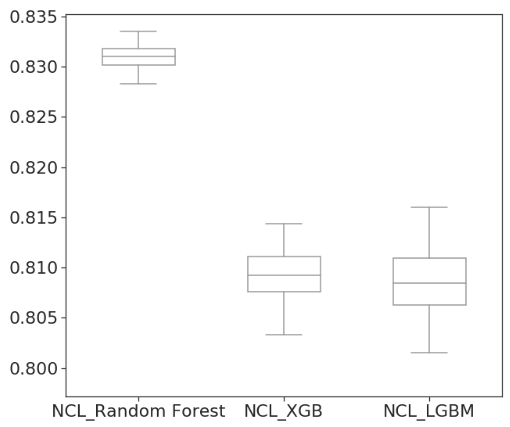
Figure C.7: Box-plots for Precision, Recall and F1-Score, when evaluating tuned models and the tuned benchmark, on the sampled *Elliptic dataset* using NCL, on the '*AF*' feature set (Experiment 2: Transactional-Level Detection).



(a) Precision

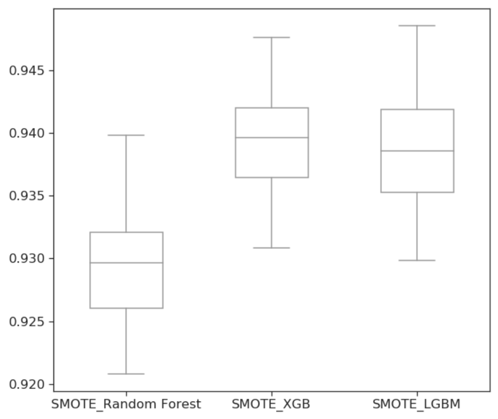


(b) Recall

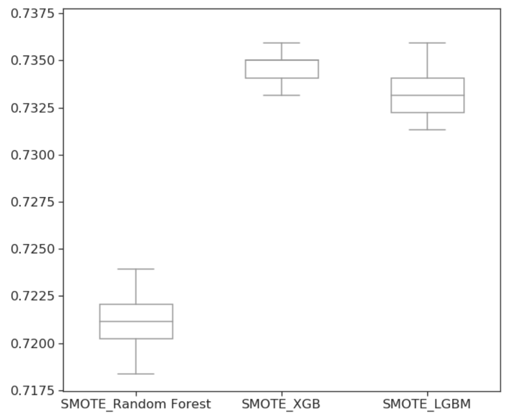


(c) F1-Score

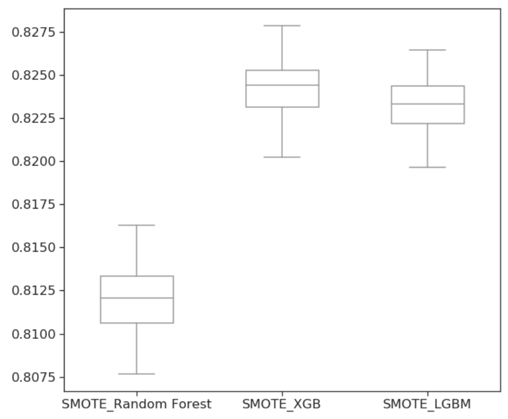
Figure C.8: Box-plots for Precision, Recall and F1-Score, when evaluating tuned models and the tuned benchmark, on the sampled *Elliptic dataset* using NCL, on the '*AF_NE*' feature set (Experiment 2: Transactional-Level Detection).



(a) Precision

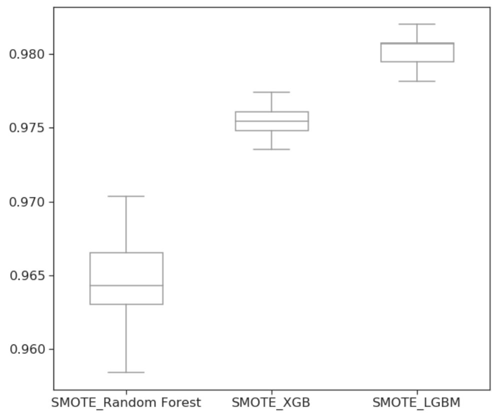


(b) Recall

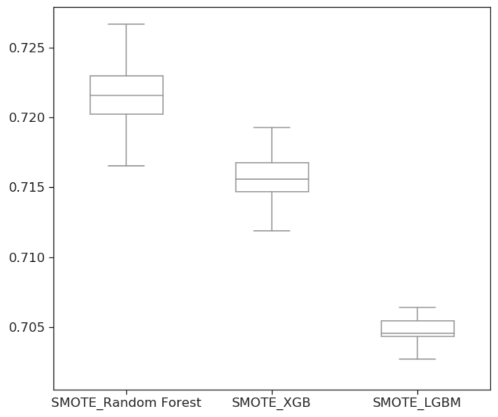


(c) F1-Score

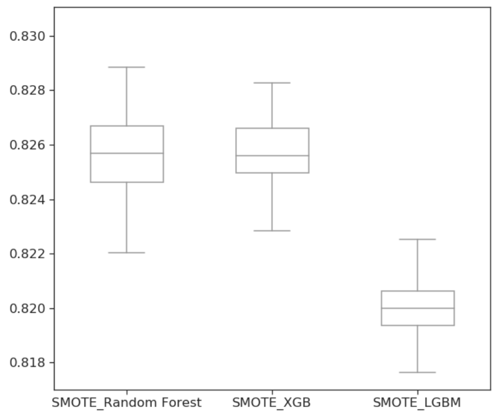
Figure C.9: Box-plots for Precision, Recall and F1-Score, when evaluating tuned models and the tuned benchmark, on the sampled *Elliptic dataset* using SMOTE, on the '*AF*' feature set (Experiment 2: Transactional-Level Detection).



(a) Precision

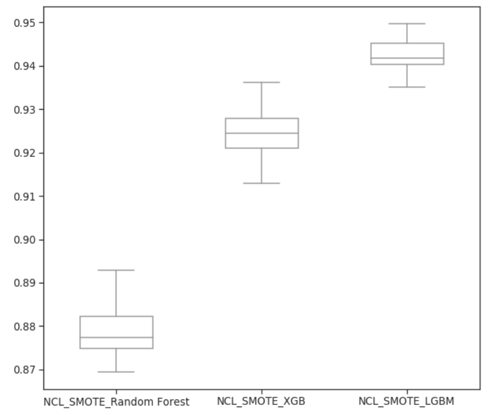


(b) Recall

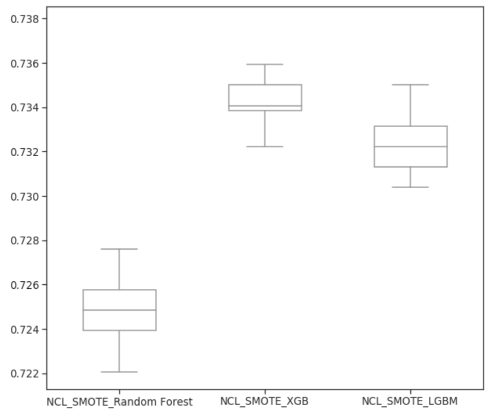


(c) F1-Score

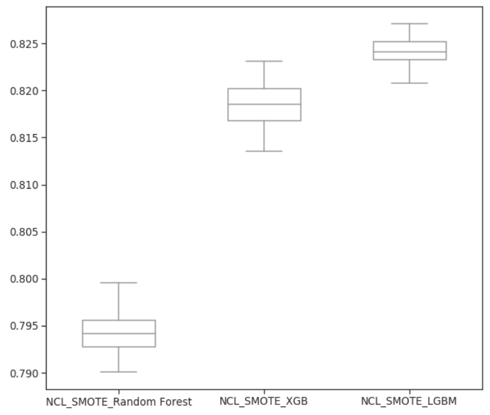
Figure C.10: Box-plots for Precision, Recall and F1-Score, when evaluating tuned models and the tuned benchmark, on the sampled *Elliptic dataset* using SMOTE, on the '*AF_NE*' feature set (Experiment 2: Transactional-Level Detection).



(a) Precision

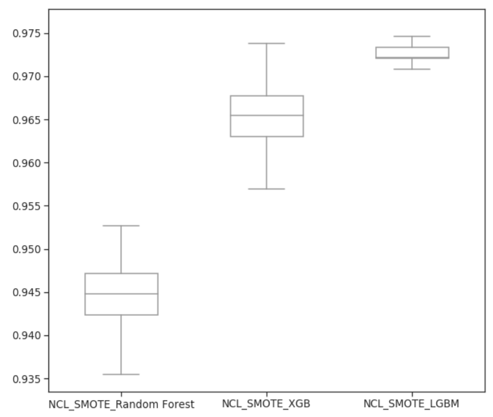


(b) Recall

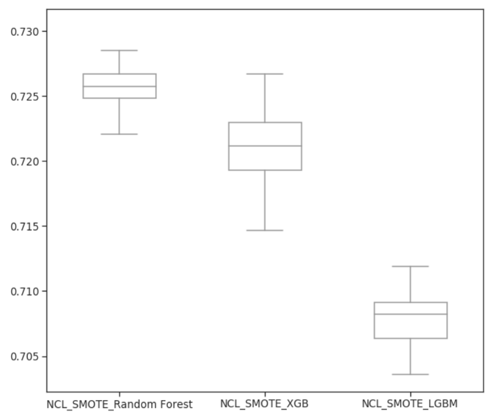


(c) F1-Score

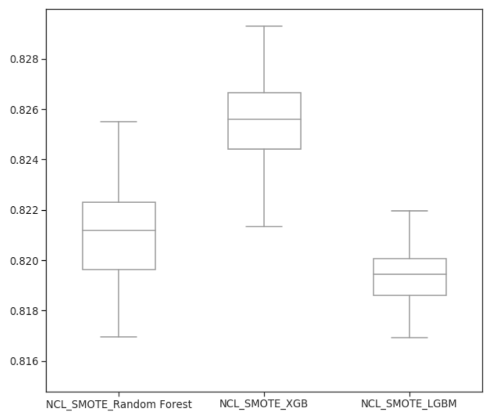
Figure C.11: Box-plots for Precision, Recall and F1-Score, when evaluating tuned models and the tuned benchmark, on the sampled *Elliptic dataset* using NCL-SMOTE, on the '*AF*' feature set (Experiment 2: Transactional-Level Detection).



(a) Precision

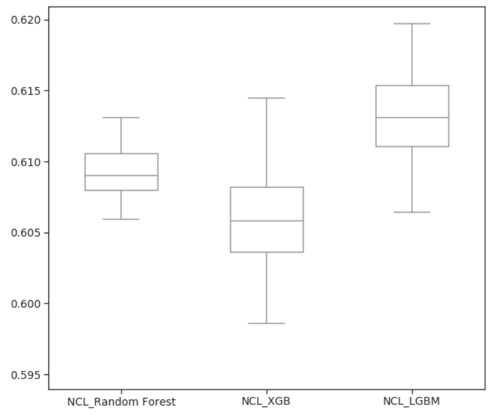


(b) Recall

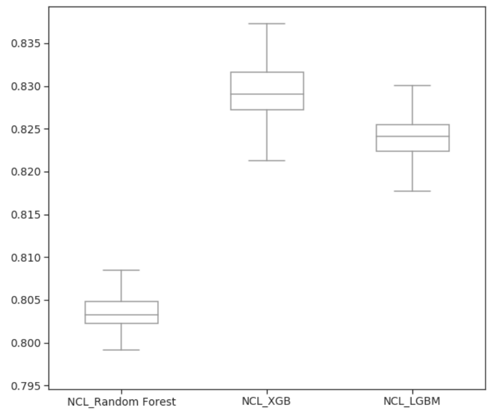


(c) F1-Score

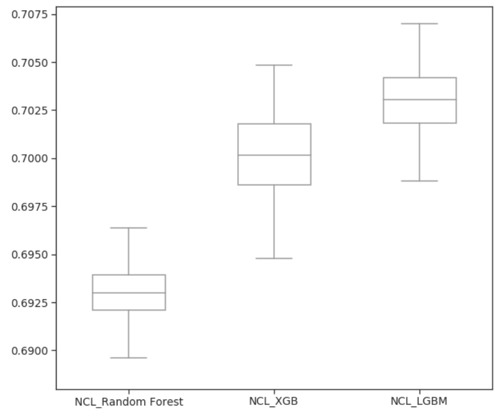
Figure C.12: Box-plots for Precision, Recall and F1-Score, when evaluating tuned models and the tuned benchmark, on the sampled *Elliptic dataset* using NCL-SMOTE, on the '*AF_NE*' feature set (Experiment 2: Transactional-Level Detection).



(a) Precision

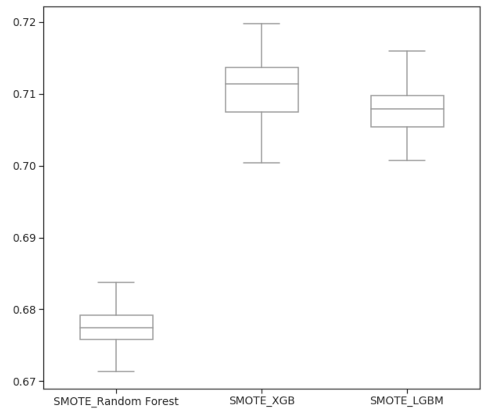


(b) Recall

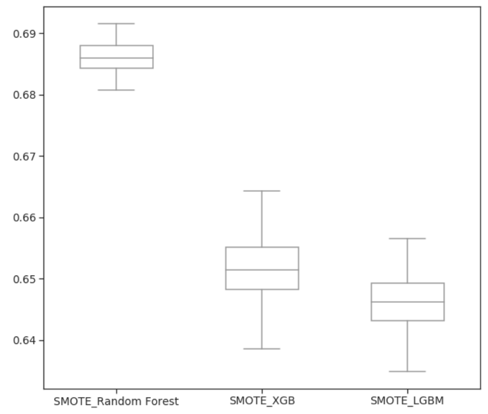


(c) F1-Score

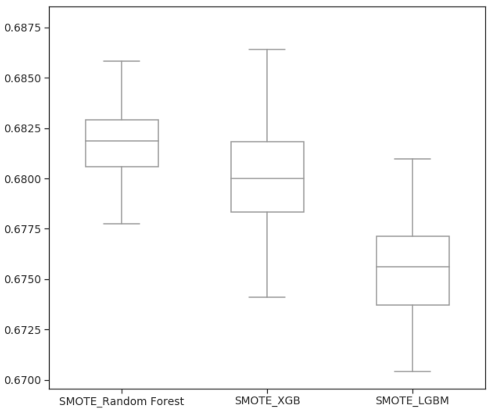
Figure C.13: Box-plots for Precision, Recall and F1-Score, when evaluating tuned models on the sampled *NOAA dataset* using NCL (Experiment 2: Supplementary Dataset).



(a) Precision

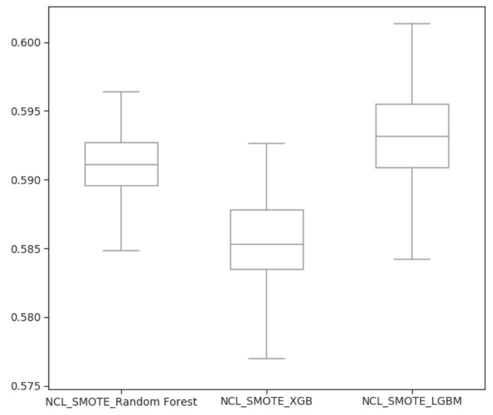


(b) Recall

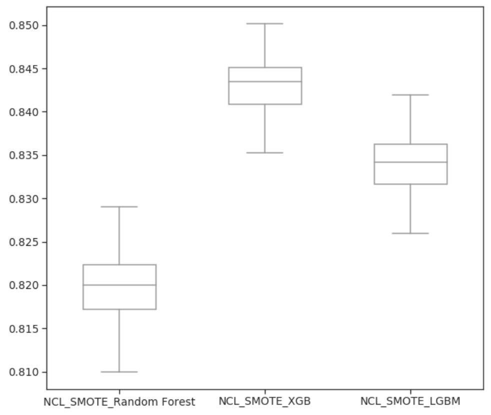


(c) F1-Score

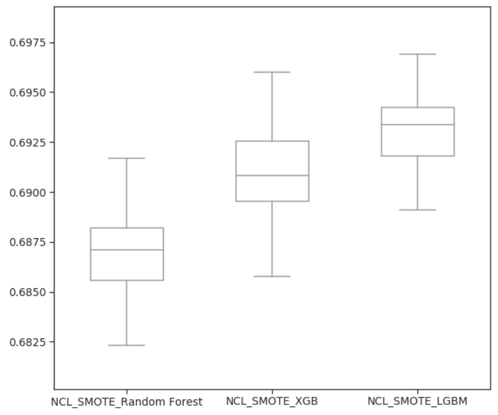
Figure C.14: Box-plots for Precision, Recall and F1-Score, when evaluating tuned models on the sampled *NOAA dataset* using SMOTE (Experiment 2: Supplementary Dataset).



(a) Precision



(b) Recall



(c) F1-Score

Figure C.15: Box-plots for Precision, Recall and F1-Score, when evaluating tuned models on the sampled *NOAA dataset* using NCL-SMOTE (Experiment 2: Supplementary Dataset).

References

Shaza M Abd Elrahman and Ajith Abraham. A review of class imbalance problem. *Journal of Network and Innovative Computing*, 1(2013):332–340, 2013.

Turner Adam and Irwin Angela Samantha Maitland. Bitcoin transactions: a digital discovery of illicit activity on the blockchain. *Journal of Financial Crime*, 25(1):109–130, 2018.

Charu C. Aggarwal. Towards systematic design of distance functions for data mining applications. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 9–18. ACM, 2003.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, KDD '19, page 2623–2631, 2019.

Aida Ali, Siti Mariyam Shamsuddin, Anca L Ralescu, et al. Classification with class imbalance problem: a review. *International Journal of Advances in Soft Computing and its Applications*, 7(3):176–204, 2015.

Alhanouf Abdulrahman Saleh Alsuwailem and Abdul Khader Jilani Saudagar. Anti-money laundering systems: a systematic literature review. *Journal of Money Laundering Control*, 2020.

Elli Androulaki, Ghassan O. Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in bitcoin. In *Financial Cryptography and Data Security*, pages 34–51. Springer, 2013.

H. Baek, J. Oh, C. Y. Kim, and K. Lee. A model for detecting cryptocurrency transactions with discernible purpose. In *2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 713–717. IEEE, 2019.

M. Bartoletti, B. Pes, and S. Serusi. Data mining for detecting bitcoin ponzi schemes. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 75–84. IEEE, 2018.

Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria C. Monard. Balancing strategies and class overlapping. In A. Fazel Famili, Joost N. Kok, José M. Peña, Arno Siebes, and Ad Feelders, editors, *Advances in Intelligent Data Analysis VI*, pages 24–35. Springer, 2005.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13:281–305, 2012.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 2546–2554, 2011.

Siddhartha Bhattacharyya, Sanjeev Jha, Kurian Tharakunnel, and J. Christopher Westland. Data mining for credit card fraud: A comparative study. *Decision Support Systems*, 50(3):602–613, 2011.

Albert Bifet and Ricard Gavalda. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining*, pages 443–448. SIAM, 2007.

Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. Leveraging bagging for evolving data streams. In *Machine Learning and Knowledge Discovery in Databases*, pages 135–150. Springer, 2010.

Albert Bifet, Eibe Frank, Geoff Holmes, and Bernhard Pfahringer. Ensembles of restricted hoeffding trees. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(2):1–20, 2012.

L. E. Boiko Ferreira, H. Murilo Gomes, A. Bifet, and L. S. Oliveira. Adaptive random forests with resampling for imbalanced data streams. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. IEEE, 2019.

D. Bratton and J. Kennedy. Defining a standard for particle swarm optimization. In *2007 IEEE Swarm Intelligence Symposium*, pages 120–127. IEEE, 2007.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Leo Breiman. Pasting small votes for classification in large databases and on-line. *Machine Learning*, 36(1): 85–103, 1999.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

Danton Bryans. Bitcoin and money laundering: mining for an effective solution. *Indiana Law Journal*, 89(1): 441–472, 2014.

Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

Chumphol Bunkhumpornpat, Krung Sinapiromsaran, and Chidchanok Lursinsap. Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem. In Thanaruk Theeramunkong, Boonserm Kijsirikul, Nick Cercone, and Tu-Bao Ho, editors, *Advances in Knowledge Discovery and Data Mining*, pages 475–482. Springer, 2009.

Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37), 2014.

Chainalysis. The 2020 state of crypto crime. `https://go.chainalysis.com/rs/503-FAP-074/images/2020-Crypto-Crime-Report.pdf`, 2020. [Online; accessed 26-May-2020].

Lakshika Sammani Chandradeva, Thushara Madushanka Amarasinghe, Minoli De Silva, Achala Chathuranga Aponso, and Naomi Krishnarajah. Monetary transaction fraud detection system based on machine learning strategies. In *Fourth International Congress on Information and Communication Technology*, pages 385–396, 2020.

David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology*, pages 199–203. Springer, 1983.

Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16(1):321–357, 2002.

C Chen, A Liaw, and L Breiman. Using random forest to learn imbalanced data. techreport 666, University of California, Berkeley, 2004.

Ke Chen, Feng-Yu Zhou, and Xian-Feng Yuan. Hybrid particle swarm optimization with spiral-shaped mechanism for feature selection. *Expert Systems with Applications*, 128:140–156, 2019.

Stephen Chen, James Montgomery, and Antonio Bolufé-Röhler. Measuring the curse of dimensionality and its effects on particle swarm optimization and differential evolution. *Applied Intelligence*, 42(3):514–526, 2015.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794.

Weili Chen, Zibin Zheng, Jiahui Cui, Edith Ngai, Peilin Zheng, and Yuren Zhou. Detecting ponzi schemes on ethereum: Towards healthier blockchain technology. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 1409–1418, 2018.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1): 21–27, 1967.

André Luis Cristiani, Tiago Pinho da Silva, and Heloisa de Arruda Camargo. A fuzzy approach for classification and novelty detection in data streams under intermediate latency. In *Brazilian Conference on Intelligent Systems*, pages 171–186. Springer, 2020.

S. Datta and A. Arputharaj. An analysis of several machine learning algorithms for imbalanced classes. In *2018 5th International Conference on Soft Computing Machine Intelligence (ISCMI)*, pages 22–27. IEEE, 2018.

A. P. Dawid. Present position and potential developments: Some personal views statistical theory the prequential approach. *Journal of the Royal Statistical Society: Series A (General)*, 147(2):278–290, 1984.

Thibault de Balthasar and Julio Hernandez-Castro. An analysis of bitcoin laundry services. In *Secure IT Systems*, pages 297–312. Springer, 2017.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7:1–30, 2006a. ISSN 1532-4435.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of machine learning research*, 7:1–30, 2006b.

Janez Demšar, Tomaž Curk, Aleš Erjavec, Črt Gorup, Tomaž Hočevar, Mitar Milutinovič, Martin Možina, Matija Polajnar, Marko Toplak, Anže Starič, Miha Štajdohar, Lan Umek, Lan Žagar, Jure Žbontar, Marinka Žitnik, and Blaž Zupan. Orange: Data mining toolbox in python. *Journal of machine learning research*, 14(1):2349–2353, 2013. ISSN 1532-4435.

G. Ditzler and R. Polikar. Incremental learning of concept drift from streaming imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 25(10):2283–2301, 2013.

Federico Divina, Aude Gilson, Francisco Goméz-Vela, Miguel García Torres, and José F Torres. Stacking ensemble learning for short-term electricity consumption forecasting. *Data Science and Big Data in Energy Forecasting*, 11(4):1–31, 2018.

Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, page 71–80, 2000.

Xibin Dong, Zhiwen Yu, Wenming Cao, Yifan Shi, and Qianli Ma. A survey on ensemble learning. *Frontiers of Computer Science*, 14(2):241–258, 2020.

Dmitry Dorofeev, Marina Khrestina, Timur Usubaliev, Aleksey Dobrotvorskiy, and Saveliy Filatov. Application of machine analysis algorithms to automate implementation of tasks of combating criminal money laundering. In *Digital Transformation and Global Society*, pages 375–385. Springer, Springer International Publishing, 2018.

Ekrem Duman and Ayse Buyukkaya. Money laundering detection using data mining. *Mining Massive Data Sets for Security: Advances in Data Mining, Search, Social Networks and Text Mining, and Their Applications to Security*, 19:287–294, 2008.

Saso Džeroski and Bernard Ženko. Is combining classifiers with stacking better than selecting the best one? *Machine learning*, 54(3):255–273, 2004.

Elliptic. Elliptic preventing and detecting criminal activity in cryptocurrencies. `https://www.elliptic.co/`, 2020. [Online; accessed 26-May-2020].

R. Elwell and R. Polikar. Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, 22(10):1517–1531, 2011.

European Parliament and Council. Fifth anti-money laundering di-rective (5amld), 2018. URL `https://eur-lex.europa.eu/eli/dir/2018/843/oj`. [Online; accessed 4-January-2020].

Yaya Fanusie and Tom Robinson. Bitcoin laundering: an analysis of illicit flows into digital currency services. *Center on Sanctions and Illicit Finance memorandum*, 2018.

Steven Farrugia, Joshua Ellul, and George Azzopardi. Detection of illicit accounts over the ethereum blockchain. *Expert Systems with Applications*, 150:113318, 2020.

Tom Fawcett. Learning from imbalanced classes. https://www.kdnuggets.com/2016/08/learning-from-imbalanced-classes.html/, 2016. [Online; accessed March 20, 2020].

L. E. B. Ferreira, J. P. Barddal, F. Enembreck, and H. M. Gomes. An experimental perspective on sampling methods for imbalanced learning from financial databases. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. IEEE, 2018.

Financial Action Task Force. International standards on combating money laundering and the financing of terrorism proliferation. the fatf recommendations, 2012. URL `"http://www.fatf-gafi.org/media/fatf/documents/recommendations/pdfs/FATF_Recommendations.pdf"`. [Online; accessed 4-May-2020].

Financial Action Task Force. Guidance for a risk-based approach to virtual assets and virtual asset service providers, 2019. URL `"http://www.fatf-gafi.org/media/fatf/documents/recommendations/RBA-VA-VASPs.pdf"`. [Online; accessed 4-May-2020].

FinCen. Application of fincen's regulations to certain business models involving convertible virtual currencies, 2019. URL `https://www.fincen.gov/sites/default/files/2019-05/FinCEN%20Guidance%20CVC%20FINAL%20508.pdf"`. [Online; accessed 4-May-2020].

Ugo Fiore, Alfredo De Santis, Francesca Perla, Paolo Zanetti, and Francesco Palmieri. Using generative adversarial networks for improving classification effectiveness in credit card fraud detection. *Information Sciences*, 479:448–455, 2019.

Eibe Frank, Geoffrey Holmes, Richard Kirkby, and Mark Hall. Racing committees for large datasets. In *Discovery Science*, pages 153–164. Springer, 2002.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139, 1997a.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997b.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer, 2001.

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

Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics Data Analysis*, 38(4):367–378, 2002.

Julie Frizzo-Barker, Peter A. Chow-White, Philippa R. Adams, Jennifer Mentanko, Dung Ha, and Sandy Green. Blockchain as a disruptive technology for business: A systematic review. *International Journal of Information Management*, 51:102029, 2020.

A. Gaihre, S. Pandey, and H. Liu. Deanonymizing cryptocurrency with graph learning: The promises and challenges. In *2019 IEEE Conference on Communications and Network Security (CNS)*, pages 1–3. IEEE, 2019.

M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera. A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):463–484, 2012.

João Gama, Indrundefined Žliobaitundefined, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4), 2014.

Heitor M. Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9):1469–1495, 2017.

Jie Gu, Yuanbing Zhou, and Xianqiang Zuo. Making class bias useful: A strategy of learning from imbalanced data. In Hujun Yin, Peter Tino, Emilio Corchado, Will Byrne, and Xin Yao, editors, *Intelligent Data Engineering and Automated Learning - IDEAL 2007*, pages 287–295. Springer, 2007.

Haibo He, Yang Bai, E. A. Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 1322–1328. IEEE, 2008.

Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. Borderline-smote: A new over-sampling method in imbalanced data sets learning. In *Advances in Intelligent Computing*, pages 878–887. Springer, 2005.

Mikkel Alexander Harlev, Haohua Sun Yin, Klaus Christian Langenheldt, Raghava Mukkamala, and Ravi Vatrapu. Breaking bad: De-anonymising entity types on the bitcoin blockchain using supervised machine learning. In *Proceedings of the 51st Hawaii International Conference on System Sciences*, pages 1–10. HICSS, 2018.

P. Hart. The condensed nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 14(3): 515–516, 1968.

Tamer Hossam Helmy, Mohamed Zaki, Tarek Salah, and Khaled Badran. Design of a monitor for detecting money laundering and terrorist financing. *Journal of Theoretical and Applied Information Technology*, 85(3): 425–436, 2016.

Juan I González Hidalgo, Bruno IF Maciel, and Roberto SM Barros. Experimenting with prequential variations for data stream learning evaluation. *Computational Intelligence*, 35(4):670–692, 2019.

Geoffrey E. Hinton. A practical guide to training restricted boltzmann machines. In Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade: Second Edition*, pages 599–619. Springer, 2012.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

Frank Hutter, Jörg Lücke, and Lars Schmidt-Thieme. Beyond manual tuning of hyperparameters. *KI - Künstliche Intelligenz*, 29(4):329–337, 2015.

Justin M. Johnson and Taghi M. Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):27, 2019.

Martin Jullum, Anders Løland, Ragnar Bang Huseby, Geir Ånonsen, and Johannes Lorentzen. Detecting money laundering transactions with machine learning. *Journal of Money Laundering Control*, 23(1):173–186, 2020.

Nutthaporn Junsomboon and Tanasanee Phienthrakul. Combining over-sampling and under-sampling techniques for imbalance dataset. In *Proceedings of the 9th International Conference on Machine Learning and Computing*, page 243–247. ACM, 2017.

Nikolei Kaplanov. Nerdy money: Bitcoin, the private digital currency, and the case against its regulation. *Loyola Consumer Law Review*, 25(1):111–174, 2012.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3149–3157, 2017.

T. M. Khoshgoftaar, C. Seiffert, J. V. Hulse, A. Napolitano, and A. Folleco. Learning with limited minority class data. In *Sixth International Conference on Machine Learning and Applications (ICMLA 2007)*, pages 348–353. IEEE, 2007.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Miroslav Kubat and Stan Matwin. Addressing the curse of imbalanced training sets: one-sided selection. In *International Conference on Machine Learning*, pages 179–186, 1997.

Salim Lahmiri, Stelios Bekiros, Anastasia Giakoumelou, and Frank Bezzina. Performance assessment of ensemble learning systems in financial data classification. *Intelligent Systems in Accounting, Finance and Management*, 27(1):3–9, 2020.

Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning*, page 473–480, 2007.

Jorma Laurikkala. Improving identification of difficult small classes by balancing class distribution. In *Proceedings of the 8th Conference on AI in Medicine in Europe: Artificial Intelligence Medicine*, page 63–66, 2001.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Chaehyeon Lee, Sajan Maharjan, Kyungchan Ko, and James Won-Ki Hong. Toward detecting illegal transactions on bitcoin using machine-learning methods. In *Blockchain and Trustworthy Systems*, pages 520–533, 2020.

Joffrey L. Leevy, Taghi M. Khoshgoftaar, Richard A. Bauder, and Naeem Seliya. A survey on addressing high-class imbalance in big data. *Journal of Big Data*, 5(1):42, 2018.

Kai Lei, Yuexiang Xie, Shangru Zhong, Jingchao Dai, Min Yang, and Ying Shen. Generative adversarial fusion network for class imbalance credit scoring. *Neural Computing and Applications*, 32(12):8451–8462, 2020.

Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(1):559–563, 2017.

Felix I Lessambo. Anti-money laundering laws. In *The US Banking System*, pages 37–66. Springer, 2020.

J. Liang, L. Li, W. Chen, and D. Zeng. Targeted addresses identification for bitcoin with network representation learning. In *2019 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 158–160. IEEE, 2019.

Jing Liao, Jianjun Zhang, and Wing WY Ng. Effects of different base classifiers to learn++ family algorithms for concept drifting and imbalanced pattern classification problems. In *2016 International conference on machine learning and cybernetics (ICMLC)*, volume 1, pages 99–104. IEEE, 2016.

Wee-Yong Lim, Amit Sachan, and Vrizlynn Thing. Conditional weighted transaction aggregation for credit card fraud detection. In *Advances in Digital Forensics X*, pages 3–16. Springer, 2014.

Y. Lin, P. Wu, C. Hsu, I. Tu, and S. Liao. An evaluation of bitcoin address classification based on transaction history summarization. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 302–310. IEEE, 2019.

Charles X. Ling and Victor S. Sheng. Cost-sensitive learning. In *Encyclopedia of Machine Learning*, pages 231–235. Springer, 2010.

V. Losing, B. Hammer, and H. Wersing. Knn classifier with self adjusting memory for heterogeneous concept drift. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 291–300. IEEE, 2016.

J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2019.

Victoria López, Alberto Fernández, Jose G. Moreno-Torres, and Francisco Herrera. Analysis of preprocessing vs. cost-sensitive learning for imbalanced classification. open problems on intrinsic data characteristics. *Expert Systems with Applications*, 39(7):6585–6608, 2012.

S. Mabunda. Cryptocurrency: The new face of cyber money laundering. In *2018 International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD)*, pages 1–6. IEEE, 2018.

James G. MacKinnon. Approximate asymptotic distribution functions for unit-root and cointegration tests. *Journal of Business & Economic Statistics*, 12(2):167–176, 1994.

James G. MacKinnon. Critical Values For Cointegration Tests. Working Paper 1227, Economics Department, Queen's University, 2010. URL `https://ideas.repec.org/p/qed/wpaper/1227.html`.

Majdi Mafarja, Ibrahim Aljarah, Hossam Faris, Abdelaziz I. Hammouri, Ala' M. Al-Zoubi, and Seyedali Mirjalili. Binary grasshopper optimisation algorithm approaches for feature selection problems. *Expert Systems with Applications*, 117:267–286, 2019.

Sebastián Maldonado and Julio López. Dealing with high-dimensional class-imbalanced datasets: Embedded feature selection for svm classification. *Applied Soft Computing*, 67:94–105, 2018.

Inderjeet Mani and I Zhang. knn approach to unbalanced data distributions: a case study involving information extraction. In *Proceedings of the ICML'03 Workshop on Learning from Imbalanced Data Sets*, 2003.

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 52(1):99–115, 1990.

Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A fistful of bitcoins: Characterizing payments among men with no names. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, page 127–140, 2013.

R. Mohammed, J. Rawashdeh, and M. Abdullah. Machine learning with oversampling and undersampling techniques: Overview study and experimental results. In *2020 11th International Conference on Information and Communication Systems (ICICS)*, pages 243–248. IEEE, 2020.

P. M. Monamo, V. Marivate, and B. Twala. A multifaceted approach to bitcoin fraud detection: Global and local outliers. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 188–194. IEEE, 2016.

Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. Scikit-multiflow: A multi-output streaming framework. *The Journal of Machine Learning Research*, 19(1):2915–2914, 2018.

Jacob Montiel, Rory Mitchell, Eibe Frank, Bernhard Pfahringer, Talel Abdessalem, and Albert Bifet. Adaptive xgboost for evolving data streams. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2020.

A. M. Mubalaike and E. Adali. Deep learning approach for intelligent financial fraud detection system. In *2018 3rd International Conference on Computer Science and Engineering (UBMK)*, pages 598–603. IEEE, 2018.

U. Mukhopadhyay, A. Skjellum, O. Hambolu, J. Oakley, L. Yu, and R. Brooks. A brief survey of cryptocurrency systems. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, pages 745–752. IEEE, 2016.

Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list*, 2009.

Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7(21): 1–21, 2013.

Andreas Isnes Nilsen. Limelight: real-time detection of pump-and-dump events on cryptocurrency exchanges using deep learning. Master's thesis, UiT Norges arktiske universitet, 2019.

N. C. Oza. Online bagging and boosting. In *2005 IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2340–2345. IEEE, 2005.

Tingting Pan, Junhong Zhao, Wei Wu, and Jie Yang. Learning imbalanced datasets based on smote and gaussian distribution. *Information Sciences*, 512:1214–1233, 2020.

The pandas development team. pandas-dev/pandas: Pandas, February 2020. URL `https://doi.org/10.5281/zenodo.3509134`.

Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B Schardl, and Charles E Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *AAAI*, pages 5363–5370, 2020.

E. S. Pearson, R. B. D'Agostino, and K. O. Bowman. Tests for departure from normality: Comparison of powers. *Biometrika*, 64(2):231–246, 1977.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 701–710, 2014.

B. Pes. Handling class imbalance in high-dimensional biomedical datasets. In *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 150–155. IEEE, 2019.

S. Phetsouvanh, F. Oggier, and A. Datta. Egret: Extortion graph exploration techniques in the bitcoin network. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 244–251. IEEE, 2018.

R. Polikar, L. Upda, S. S. Upda, and V. Honavar. Learn++: an incremental learning algorithm for supervised neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 31 (4):497–508, 2001.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: Unbiased boosting with categorical features. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 6639–6649, 2018.

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993. URL `http://portal.acm.org/citation.cfm?id=152181`.

Jesse Read, Albert Bifet, Bernhard Pfahringer, and Geoff Holmes. Batch-incremental versus instance-incremental learning in dynamic and evolving data. In *International symposium on intelligent data analysis*, pages 313–323. Springer, 2012.

Daniel Rodriguez, Israel Herraiz, Rachel Harrison, Javier Dolado, and José C. Riquelme. Preliminary comparison of techniques for dealing with imbalance in software defect prediction. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14. ACM, 2014.

Lior Rokach. Decision forest: Twenty years of research. *Information Fusion*, 27:111–125, 2016.

Neelam Rout, Debahuti Mishra, and Manas Kumar Mallick. Handling imbalanced data: A survey. In M. Sreenivasa Reddy, K. Viswanath, and Shiva Prasad K.M., editors, *International Proceedings on Advances in Soft Computing, Intelligent Systems and Applications*, pages 431–443. Springer, 2018.

Sankardas Roy, Jordan DeLoach, Yuping Li, Nic Herndon, Doina Caragea, Xinming Ou, Venkatesh Prasad Ranganath, Hongmin Li, and Nicolais Guevara. Experimental study with real-world data for android app security analysis using machine learning. In *Proceedings of the 31st Annual Computer Security Applications Conference*, ACSAC 2015, page 81–90, 2015.

Omer Sagi and Lior Rokach. Ensemble learning: A survey. *WIREs Data Mining and Knowledge Discovery*, 8 (4):e1249, 2018.

S. Samanta, B. K. Mohanta, S. P. Pati, and D. Jena. A framework to build user profile on cryptocurrency data for detection of money laundering activities. In *2019 International Conference on Information Technology (ICIT)*, pages 425–429. IEEE, 2019.

David Savage, Qingmai Wang, Xiuzhen Zhang, Pauline Chou, and Xinghuo Yu. Detection of money laundering groups: Supervised learning on small networks. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Ernesto Ugo Savona and Michele Riccardi. Assessing the risk of money laundering: research challenges and implications for practitioners. *European Journal on Criminal Policy and Research*, 25(1):1–4, 2019.

Gehad Ismail Sayed, Alaa Tharwat, and Aboul Ella Hassanien. Chaotic dragonfly algorithm: an improved metaheuristic algorithm for feature selection. *Applied Intelligence*, 49(1):188–205, 2019.

Robert E. Schapire. Using output codes to boost multiclass learning problems. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, page 313–321, 1997.

Lothar M. Schmitt. Theory of genetic algorithms. *Theoretical Computer Science*, 259(1):1–61, 2001.

Friedrich Schneider and Ursula Windischbauer. Money laundering: some facts. *European Journal of Law and Economics*, 26(3):387–404, 2008.

Paul Allan Schott. *Reference guide to anti-money laundering and combating the financing of terrorism*. The World Bank, 2006.

David Schwartz, Noah Youngs, Arthur Britto, et al. The ripple protocol consensus algorithm. *Ripple Labs Inc White Paper*, 5(8), 2014.

C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano. Rusboost: A hybrid approach to alleviating class imbalance. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 40(1): 185–197, 2010.

Ted E Senator, Henry G Goldberg, Jerry Wooton, Matthew A Cottini, AF Umar Khan, Christina D Klinger, Winston M Llamas, Michael P Marrone, Raphael WH Wong, et al. The fincen artificial intelligence system: Identifying potential money laundering from reports of large cash transactions. In *IAAI*, pages 156–170, 1995.

Kishore Singh and Peter Best. Anti-money laundering: Using data visualization to identify suspicious activity. *International Journal of Accounting Information Systems*, 34:100418, 2019.

Akila Somasundaram and Srinivasulu Reddy. Parallel and incremental credit card fraud detection model to handle concept drift and data imbalance. *Neural Computing and Applications*, 31(1):3–14, 2019.

W. Nick Street and YongSeog Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, page 377–382, 2001.

Agus Sudjianto, Sheela Nair, Ming Yuan, Aijun Zhang, Daniel Kern, and Fernando Cela-Díaz. Statistical methods for fighting financial crimes. *Technometrics*, 52(1):5–19, 2010.

Hao Hua Sun Yin, Klaus Langenheldt, Mikkel Harlev, Raghava Rao Mukkamala, and Ravi Vatrapu. Regulating cryptocurrencies: a supervised machine learning approach to de-anonymizing the bitcoin blockchain. *Journal of Management Information Systems*, 36(1):37–73, 2019.

Melanie Swan. *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015.

Zbigniew Tarapata, Rafal Kasprzyk, and Kamil Banach. Graph-network models and methods used to detect financial crimes with iafec graphs it tool. In *22nd International Conference on Circuits, Systems, Communications and Computers (CSCC 2018)*, volume 210, pages 1–6. EDP Sciences, 2018.

I. Tomek. Two modifications of cnn. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(11):769–772, 1976.

K. Toyoda, T. Ohtsuki, and P. T. Mathiopoulos. Identification of high yielding investment programs in bitcoin via transactions pattern analysis. In *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pages 1–6. IEEE, 2017.

K. Toyoda, T. Ohtsuki, and P. T. Mathiopoulos. Multi-class bitcoin-enabled service identification based on transaction history summarization. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1153–1160. IEEE, 2018.

Lawrence J Trautman. Virtual currencies; bitcoin & what now after liberty reserve, silk road, and mt. gox? *Richmond Journal of Law and Technology*, 20(4):1–108, 2014.

Shivani Tyagi and Sangeeta Mittal. Sampling approaches for imbalanced data classification problem in machine learning. In *Proceedings of ICRIC 2019*, pages 209–221. Springer, 2020.

UNODC. Estimating illicit financial flows resulting from drug trafficking and other transnational organized crimes. *National Criminal Justice Reference Service*, 2011.

Nadja van der Veer. *Money Laundering Laws, Technology and Keeping Up With Criminals*, chapter 28, pages 94–96. Wiley, 2019.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, Aditya Vijaykumar, Alessandro Pietro Bardelli, Alex Rothberg, Andreas Hilboll, Andreas Kloeckner, Anthony Scopatz, Antony Lee, Ariel Rokem, C. Nathan Woods, Chad Fulton, Charles Masson, Christian Häggström, Clark Fitzgerald, David A. Nicholson, David R. Hagen, Dmitrii V. Pasechnik, Emanuele Olivetti, Eric Martin, Eric Wieser, Fabrice Silva, Felix Lenders, Florian Wilhelm, G. Young, Gavin A. Price, Gert-Ludwig Ingold, Gregory E. Allen, Gregory R. Lee, Hervé Audren, Irvin Probst, Jörg P. Dietrich, Jacob Silterra, James T. Webber, Janko Slavič, Joel Nothman, Johannes Buchner, Johannes Kulick, Johannes L. Schönberger, José Vinícius de Miranda Cardoso, Joscha Reimer, Joseph Harrington, Juan Luis Cano Rodríguez, Juan Nunez-Iglesias, Justin Kuczynski, Kevin Tritz, Martin Thoma, Matthew Newville, Matthias Kümmerer, Maximilian Bolingbroke, Michael Tartre, Mikhail Pak, Nathaniel J. Smith, Nikolai Nowaczyk, Nikolay Shebanov, Oleksandr Pavlyk, Per A. Brodtkorb, Perry Lee, Robert T. McGibbon, Roman Feldbauer, Sam Lewis, Sam Tygier, Scott Sievert, Sebastiano Vigna, Stefan Peterson, Surhud More, Tadeusz Pudlik, Takuya Oshima, Thomas J. Pingel, Thomas P. Robitaille, Thomas Spura, Thouis R. Jones, Tim Cera, Tim Leslie, Tiziano Zito, Tom Krauss, Utkarsh Upadhyay, Yaroslav O. Halchenko, Yoshiki Vázquez-Baeza, and SciPy 1.0 Contributors. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature Methods*, 17(3):261–272, 2020.

Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, page 226–235, 2003.

S. Wang and X. Yao. Diversity analysis on imbalanced data sets by using ensemble models. In *2009 IEEE Symposium on Computational Intelligence and Data Mining*, pages 324–331. IEEE, 2009.

Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I Weidele, Claudio Bellei, Tom Robinson, and Charles E Leiserson. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. 2019.

Gary M. Weiss. Mining with rarity: A unifying framework. *ACM Sigkdd Explorations Newsletter*, 6(1):7–19, 2004.

Gary M Weiss, Kate McCarthy, and Bibi Zabar. Cost-sensitive learning vs. sampling: Which is best for handling unbalanced classes with unequal error costs? *Dmin*, 7(35-41), 2007.

Wes McKinney. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.

R. H. Wilcox. Adaptive control processes—a guided tour. *Naval Research Logistics Quarterly*, 8(3):315–316, 1961.

D. L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-2(3):408–421, 1972.

Russ Wolfinger and Pei yi Tan. Stacked ensemble models for improved prediction accuracy. In *Proc. Static Anal. Symp.*, pages 1–19, 2017.

David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

Yufei Xia, Chuanzhe Liu, YuYing Li, and Nana Liu. A boosted decision tree approach using bayesian hyper-parameter optimization for credit scoring. *Expert Systems with Applications*, 78:225–241, 2017.

Kunlin Yang and Wei Xu. Fraudmemory: Explainable memory-enhanced sequential neural networks for financial fraud detection. In *Proceedings of the 52nd Hawaii International Conference on System Sciences*, pages 1023–1032, 2019.

Peter Yeoh. Banks' vulnerabilities to money laundering activities. *Journal of Money Laundering Control*, 23 (1):122–135, 2019.

Huang Jimmy Yicheng. Effectiveness of us anti-money laundering regulations and hsbc case study. *Journal of Money Laundering Control*, 18(4):525–532, 2015.

Yong Sun and Feng Liu. Smote-ncl: A re-sampling method with filter for network intrusion detection. In *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, pages 1157–1161. IEEE, 2016.

Yong Yuan and Fei-Yue Wang. Blockchain and cryptocurrencies: Model, techniques, and applications. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(9):1421–1428, 2018.

G Zames, NM Ajlouni, NM Ajlouni, NM Ajlouni, JH Holland, WD Hills, and DE Goldberg. Genetic algorithms in search, optimization and machine learning. *Information Technology Journal*, 3(1):301–302, 1981.

C. Zhang, G. Wang, Y. Zhou, L. Yao, Z. L. Jiang, Q. Liao, and X. Wang. Feature selection for high dimensional imbalanced class data based on f-measure optimization. In *2017 International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)*, pages 278–283, 2017.

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

Indrė Žliobaitė, Mykola Pechenizkiy, and João Gama. An overview of concept drift applications. In *Big Data Analysis: New Algorithms for a New Society*, pages 91–114. Springer, 2016.

F. Zola, M. Eguimendia, J. L. Bruse, and R. Orduna Urrutia. Cascading machine learning to attack bitcoin anonymity. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 10–17. IEEE, 2019.

Francesco Zola, Jan Lukas Bruse, Maria Eguimendia, Mikel Galar, and Raul Orduna Urrutia. Bitcoin and cybersecurity: temporal dissection of blockchain data to unveil changes in entity behavioral patterns. *Applied Sciences*, 9(23):5003, 2019.