PUBLIC UNIVERSITY OF NAVARRE

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

DEPARTMENT OF STATISTICS, COMPUTER SCIENCE AND MATHEMATICS

Behavioral analysis in Cybersecurity using Machine Learning. A study based on graph representation, class imbalance and temporal dissection

Francesco Zola

DOCTORAL THESIS

Pamplona, June 2022

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

https://doi.org/10.48035/Tesis/2454/44259

PUBLIC UNIVERSITY OF NAVARRE

upna

Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

DEPARTMENT OF STATISTICS, COMPUTER SCIENCE AND MATHEMATICS

Behavioral analysis in Cybersecurity using Machine Learning. A study based on graph representation, class imbalance and temporal dissection

Thesis presented by

Francesco Zola

to qualify for the PhD degree at

Public University of Navarre

Pamplona, June 2022

Acknowledgements

Un camino durato tre anni pieno di insidie, momenti belli e brutti, attimi di gioia e di frustazione. Sono stati tre anni intensi divisi tra gli obiettivi del lavoro e quelli personali, con la consapevolezza di potercela fare e di poter arrivare a questo splendido traguardo. In tutto questo tempo, la passione per il lavoro e la voglia di mettersi in gioco (aiutato dalla mia testardaggine), mi hanno portato a non mollare mai e dare tutto me stesso, fino a diventare la persona che sono oggi. Guardandomi indietro, non posso che essere soddistatto di tutti i traguardi raggiunti in questi anni siano essi a livello lavorativo ma anche a livello umano e affettivo.

Inanzitutto, vorrei ringrazie Margherita, amica, fidanzata ed ora moglie che mi ha sempre su(o)pportato in tutti questi anni. Grazie per aver creduto sempre in me ed aver assecondato molte mie follie, come quella di lasciare tutto e partire per San Sebastian. Un ringraziamento speciale va ai miei genitori Ada e Vincenzo, anche se a 1.800km di distanza, sono sempre stati presenti. Loro mi hanno insegnato che nella vita nulla è dovuto, e che bisogna lavorare ed impegnarsi seriamente per ottenere ció che veramente si desidera. Insieme a loro vorrei ringraziare mia sorella Chiara e la sua splendida famiglia, grazie a Helena e Nathan che mi hanno riempito la vita di molteplici gioie.

Sentimentalismi a parte, un enorme grazie va ai miei 2 (+1) tutori, a coloro grazie ai quali é stato possible completare questo percorso, che mi hanno arrichito professionalmente e personalmente, mi hanno spronato a dare il meglio, a volte anche con revisioni molto dure. É stato un onore lavorare con il Dr. Jan Bruse ed il Dr. Mikel Galar, grazie per la vostra infinita pazienza ed il vostro immenso aiuto. Un ringraziamento speciale va al Dr. Raul Orduna che, anche senza un coinvolgimento “ufficiale”, ha manovrato tutto nell’ombra, e mi ha guidato in questo lungo cammino.

Vorrei inoltre ringraziare il prof. Dr. Lorenzo Cavallaro per avermi dato l’opportunitá di collaborare con l’University College of London (UCL) ed il prof. Dr. Zeno J.M.H. Geradts per avermi accolto nel Netherlands Forensic Institute (NFI). Entrambe sono state due grandissime esperienze formative che mi ha permesso di vedere il mondo della ricerca da diverse prospettive e mi hanno aiutato a dare un impronta internazionale al mio lavoro.

C’e un ultimo gruppo di pazzi scalmanati che desidero ringraziare per avermi accompagnato in questo percorso, parlo di Jon, Lander, Xabi, Telmo, Alvaro, Maria, Juan, Idoia, David, Roberto, Igor, Oscar, Ines, e molti altri (sarebbe impossible elencarli tutti), con i quali ho condiviso progetti, publicazioni, riunioni, preoccupazioni, ma anche pintxos, area days, manga, feste e molto altro.

Con loro il lavoro é sembrato meno faticoso ed é stato molto piú piacevole. Per finire, un doveroso ringraziamento va anche al centro di ricerca Vicomtech, che mi ha dato l'opportunitá di conseguire questo splendido traguardo.

Grazie mille a tutti, anche a coloro che non ci sono piú, e che hanno lasciato un segno indelebile nella mia vita. Grazie per avermi accompagnato in questo meraviglioso viaggio.

GRAZIE MILLE! GRACIAS A TODOS!

Eskerrik asko guztioi!

Index

Part I. Thesis

1. Introduction

Nowadays, cybersecurity plays a key role in everyday life. Each day more and more devices are interconnected sharing information, communications, and connections. Paradigms like IoT, Industry 4.0, Smart Factory, 5G, have increased the velocity of such growth. According to [LPS21], there are about 20 billion devices connected to the global net, like wearables, medical devices, automotive control units, smartphones, televisions, fridges, and so on. In this scenario, the attack surface [ROC+20] has increased enhancing the number of possible threats and vulnerabilities that can be exploited by a cyber-attack. A cyber-attack can be seen as an attack on the Confidentiality, Availability, and Integrity (CIA triad) of an information system, network, software, etc. [SC14]. The consequences of a cyber-attack are not only limited to digital leaks or information losses, but can have a very huge impact in economical, ethical, digital, and societal terms for the attacked company. For this reason, data protection and the management/monitoring of information systems has become a primary task for many companies [MPDH19], as well as for the European Community.

Cybersecurity researchers and experts have started to exploit new paradigms and technologies for increasing the ability to detect threats and vulnerabilities. This is the case of Machine Learning (ML) [XKL+18], a branch of Artificial Intelligence (AI) that helps to build automatic learning models, i.e., systems able to learn specific behaviors from a training dataset. Then, these systems, also known as models, can be used to perform predictions over previously unseen data. Machine learning, and derived techniques such as Deep Learning (DL), have shown to be promising solutions for discovering new insights from data and for quantifying cyber risks [SKB+20].

ML models can be divided into several groups according to their learning process or the signal used as input. In particular, the main categories are *supervised, unsupervised* and *reinforcement learning*. However, in this thesis, we focus on the first two categories, which are introduced and further discussed. Supervised learning is a ML approach in which algorithms are trained with labeled data, i.e., the used dataset has information about the desired target [KZP+07]. In this way, during the learning process, the model is driven for reaching the desired output. These models are mainly used for classification or regression tasks. In unsupervised learning, the training dataset has no information about the desired output, so the models are trained for searching similarities and patterns among the input data [UQR+19]. These models are mainly used for clustering, dimensionality reduction, and anomaly detection. Notably, among the unsupervised approaches, *Generative Adversarial Networks (GANs)* have shown an increasing interest, since they allow one to learn the input distribution making possible to generate realistic synthetic samples. GANs are composed of two networks that are trained simultaneously via an adversarial process. This competition during the training process allows the networks to improve their abilities and their final performances [GPAM+14]

ML algorithms are mainly applied in the cybersecurity domain to analyze the input data as time series [ZCFM06, CKBR06]. In fact, when we talk about events, communications, logs, and

transactions, it is common to analyze the temporal relation between the data. This temporal characteristic is very relevant since it allows one to evaluate and predict immediate threats and take appropriate decisions. In fact, based on the threat/vulnerability prediction, it is possible to propose strategic and operational decisions to operators/experts in order to mitigate the possible problem, facilitate backup, disaster recovery, diversity planning, maintenance scheduling, etc. [PRT+17]. Although this approach is well-suited for real-time systems and critical infrastructures, i.e., applications where timing is fundamental in order to quickly apply countermeasures and reduce the impact of a cyber-attack, it is also true that this time series analysis only provides a partial view of what is going on. This partial information can produce skewed results characterized by a high number of false positive elements [LAF15], which can consequently lead to take inappropriate actions over the environment, compromising its usability and its performance. Furthermore, in complex systems, entity interactions can evolve, disappear, emerge or simply change their dynamics and so time-series information is not enough for adequately addressing classification or anomaly detection tasks.

As shown in previous works [ATK15,LJRH11], a solution to these problems can be implemented by exploiting graph-based structures. These graphs promote the integration of both structured and unstructured data highlighting entities (nodes) and their relationships (edges), allowing the representation and definition of more complex behaviors. This approach allows getting a wider vision of the problem, generating complementary results to the ones obtained by exploiting directly the time series. In particular, using graph-based structures, the evaluation is not only performed on the single connection, transaction, or event, but it is possible to evaluate the behavior of the entity that has generated it, allowing to take more appropriate actions. Inspired by their promising results and potentialities, which have been little explored in the cybersecurity domain, in this thesis, graph-based structures are considered.

These graph representations can be used for node, link, or graph prediction/classification. Node classification is a common task on graph data in which the goal is to assign a category/label to each node of the graph. For example, this operation is used for analyzing users' behaviors in social networks [LPL21], or for detecting botnets [CKA+17]. Link prediction is a task that can be used to predict both a category/label of an existing edge or a "missing" edge between nodes. This operation is used, for example, for recommending content to users on social platforms [Sch14] or predicting drug side effects [AAD+21]. Graph classification is a task used for analyzing the whole graph structure (topology) and for determining a category/label for the graph. This is the case of molecule toxicity [JWW21] or computer program analysis [YYJ19].

However, as mentioned before, in cybersecurity applications, the available information is inherently stored as time series or sequential data, so for creating the graph structures, three main questions need to be addressed.

1. *How should the nodes be defined?*

2. *What should the edges connecting the nodes represent?*

3. *How is the behavior of a node modelled?*

For example, using a network traffic dataset, it is possible to define nodes as different internet protocol (IP) addresses, media access control (MAC) addresses, or even services. At the same time, edges can be used for representing packets, traffic flows, routine calls, byte flows, etc., whereas the behaviors can be computed by evaluating the nodes' interactions within the obtained graph. An example of this approach is introduced in [IPF+07] and [JSZ09], where Traffic Dispersion Graph (TDG) and Traffic Activity Graphs (TAGs) are described, respectively. However, these definitions can change according to the input data used, for example, to analyze transaction networks address-transaction graphs can be used [FKP15], for social networks or mobile phone data MultiAspect Graphs (MAGs) [WFZ16] or Time-Varying Graphs (TVG) [CFQS12] can be considered, while for program binary data, Abstract Syntax Trees (ASTs) and Control Flow Graphs (CFGs) can be applied [PRP21]. Hence, the process of graph creation strongly depends on the application domain and the goal to be achieved. In particular, in this thesis, we focus the analysis on three specific cybersecurity applications, two based on node classification using network traffic data and crypto-transactions, and the third one based on graph classification using malware binaries.

In several applications, data temporality can affect the graph representation [GCC20,HML+14]. This is the case of node classification using network traffic data and crypto-transactions in which we have network behaviors evolving continuously. In these scenarios, the temporal aspect plays a key role and should be analyzed too. In fact, considering the whole dataset at once for creating a unique static and monolithic graph can require too much computational effort making the real application of the model complex and heavy. Furthermore, the created graph can get overloaded with information, where key aspects may be unnoticed. Inspired by [IFM09], which extends the concept of TDG by introducing a temporal TDG, in this thesis, when node classification problems are analyzed, we propose and validate temporal dissection approaches that try to deal with the data temporality. On the other hand, in cases in which the goal is to classify the entire graph structure, as in the case of malware binaries, the temporal aspect is not directly involved in the graph creation, since every structure is defined in a static file. In this sense, temporality should be studied in how the properties of malware binaries evolve over time.

Once the graphs representations are defined, and eventual temporal concerns addressed, we propose to use ML models for the final classification. In this sense, two main approaches are commonly used when the information is represented as a graph. On the one hand, it is possible to extract characteristics (also called *features* or *embeddings*) that represent the entire graph or each node (according to the classification goal) [CZC18], and then use this information for training and testing traditional ML models. In this way, relational information is not directly analyzed, since the input of the model is a numeric vector of characteristics. In this thesis, we call these models *no-relational*. On the other hand, as introduced in [SGT+08, KW17], it is possible to apply DL paradigms directly over the graphs with the aim to facilitate the training process and enhance the model performance in discovering network patterns. In this way, the graph relations are directly involved in the classification tasks. We referred to them as *relational* models. In this thesis, we implement and explore both relational and no-relational models depending on the specific application, and in particular, we directly compare them in a behavioral classification task using network data.

The classification performance of a model strongly depends on the data used during the training process. For this reason, the used dataset needs to be analyzed to detect eventual bias or imbalance problems that can skew the learning process. In fact, in many cybersecurity applications, as well as in many other domains, datasets are often characterized by a highly non-homogeneously population, which means that some classes present in the dataset are more populated than others, causing a class imbalance problem [FGG+18b]. This is because it is easier to find information about normal activities rather than about "rare" ones. This phenomenon can strongly affect the quality of the classification especially when supervised ML techniques are used [JS02]. More specifically, when graph-based structures are involved, traditional data-level preprocessing techniques [FGG+18b], as well as graph sampling techniques based on a node or edge sampling [WCA+16], cannot be used without altering the graph topology and consequently, the problem description, whereas graph transversal sampling techniques [HL13] show limitations with disconnected graphs. For this reason, in this thesis, we focus the analysis on addressing imbalance problems directly over disconnected graphs, avoiding the changes in the graph components' topology. On the other hand, we analyze the cybersecurity imbalance problem also from a different point of view, proposing an unsupervised approach based on GAN for generating new synthetic data that can be used for balancing the initial dataset. In particular, this approach is not validated directly using graph structures, but using tabular data that represent node behaviors extracted from graphs.

Usually, in cybersecurity applications, ML experts and researchers are mainly focused on analyzing and evaluating the model performance [UAB19,SH18]. In this sense, they try to exploit different data structures such as time-series or graphs, address class imbalance problems, clean noisy data, and so on, with the main goal to improve the classification performance. However, they neglect the interpretation of the results, which is a fundamental task for extracting more knowledge about the input data, model behavior, and their evolution over time. This task becomes even more relevant in the cybersecurity domain, since operators and analysts are not ML experts. Hence, they see the trained ML models as "black boxes", in which input data is converted into a classification prediction without understating the motivation behind the decision. This situation favors generic pitfalls and assumptions that can lead to over-optimistic results, conclusions, and lessons learned [AQP+22]. For this reason, for addressing these concerns, in this thesis, we propose a Cyber Threat Intelligence (CTI) approach which helps to switch the focus from the classifier performance to the feature trends and drift in order to resume the control over the problem. CTI represents a crucial step for evaluating and learning more information about the data evolution and the classifier behavior [CDD18]. This information can be then used for enhancing the analyst's expertise and for improving the detection of future attacks [DCD+18].

In summary, in this thesis, we present the analyses, proposals, and results obtained by applying graph representation for behavioral classification in the cybersecurity domain. More specifically, we propose to carefully consider the temporal aspect that can affect the graph creation as well as the usability and performance of the final classifier. Then, we introduce two novel approaches that can be used for addressing the class imbalance problem directly over the graph-based structures. At the same time, we explore other novel techniques that have already shown good results in other domains, like the ones based on GANs, that need adaptations to be used with cybersecurity

behavioral data. Nevertheless, these techniques do not work directly over the graph but over data extracted from graph structures. Finally, we propose an extensive analysis for extracting additional insights from trained models for enhancing CTI. The idea of this analysis is not to focus the attention just on the performance results, but to highlight trends and behaviors that can generate performance drift and reduce the reliability of the model.

To accomplish these objectives the thesis is divided into two parts:

- Part I. Includes the problem statement, motivation and objectives, as well as the main results, discussion and final conclusions
- Part II. Contains the publications associated with this thesis.

In Part I, after a brief introduction, general concepts about cybersecurity, graphs, class imbalance, and GANs are introduced in Section 2. In Section 3, the motivation of this thesis is presented, whereas in Section 4 the objectives are discussed. Afterward, in Section 5, we summarize the works carried out along this dissertation as well as the discussion about the main results. Finally, conclusions are drawn in Section 6, whereas possible improvements and future works are analyzed in Section 7.

In Part II, the four publications related to the thesis are provided:

- *Bitcoin and cybersecurity: Temporal dissection of blockchain data to unveil changes in entity behavioral patterns*
- *Network traffic analysis through node behaviour classification: a graph-based approach with temporal dissection and data-level preprocessing*
- *Attacking Bitcoin anonymity: Generative Adversarial Networks for improving Bitcoin entity classification*
- *Cyber Threat Intelligence for Malware Classification in the Presence of Concept Drift*

2. Problem Statement

Today, there are about 20 billion devices connected to the global net [LPS21], which belong to companies across multiple sectors, from Information and Communications Technology (ICT) to medical, automotive to energy plants, transportation, etc. Furthermore, mobile phones, tablets, wearables, smart TVs, and even house lights have brought private users' information into the loop [TBS+18]. In this sense, the consequences of a cyber-attack can impact economical, social, and human aspects (in terms of safety, privacy, reputation, or psychology) [ANG+18]. This novel scenario has increased cybersecurity concerns in private and public entities, companies, and communities. More specifically, the European Union (EU) is leading efforts to improve the resilience to cyber threats and to reduce the risks associated with cybersecurity [1].

[1] https://digital-strategy.ec.europa.eu/en/policies/cybersecurity-strategy

This growth in the device interconnections makes also possible to gather a huge amount of information that can be used by Law Enforcement Agencies (LEAs), analysts, and operators for forensic investigation, detecting cybercrimes, money-laundering monitoring, fighting counter-terrorism, cross-border tracking, and so on [Int21]. However, such a large amount of data slows down investigations, since LEAs, analysts, and operators need to spend a lot of time on data processing tasks. In this sense, new paradigms such as Artificial Intelligence (AI), Machine Learning (ML), and Cyber Threat Intelligence (CTI) can be exploited to speed up these tasks allowing one to react faster and manage the time and effort for each investigation more effectively.

For this reason, in this thesis, we investigate how to combine ML algorithms, graph representation, and temporal dissection for cybersecurity behavioral analysis. More specifically, on the one hand, we propose to analyze how behaviors evolve over time, which can be a hint of malicious or anomaly traffic. On the other hand, we propose to address common ML problems such as class imbalance, for improving the model performance and so the quality of the results. Finally, we exploit CTI concepts for drawing a methodology that can be used in cybersecurity applications for extracting novel knowledge and helping operators and analysts in their tasks.

To guide the reader during this thesis, we first need to clarify several general aspects of cybersecurity, the taxonomy considered, and which are the most relevant cybersecurity task (Section 2.1). Then, since this thesis proposes the usage of graph-based structure, concepts related to graph theory, techniques used for graph representation, and graph machine learning are described (Section 2.2). In Section 2.3, the class imbalance problem and traditional techniques used for addressing it in tabular and graph-based data are recalled. Finally, in Section 2.4, an overview of the GAN architecture, possible downside effects, and its common applications are reported, whereas in Section 2.5 several concepts related to CTI are described.

2.1. Cybersecurity

According to the standard ISO/IEC 27000[2], a cyber-attack can be defined as *"an attempt to destroy, expose, alter, disable, steal or gain unauthorized access to or make unauthorized use of anything that has value to the organization"*. Furthermore, when a cyber-attack is deployed violating the law, we are in presence of a cybercrime [SRP12]. Usually, cybercrimes are carried out for profit purposes, however, they can be also used for directly damaging a system, smuggling child pornography and intellectual property, stealing an identity, violating the privacy, etc. In this sense, cybersecurity tries to apply defense mechanisms (countermeasures) for protecting any systems, services, networks, and data from cyber-attacks. These defense mechanisms are implemented to preserve the three pillars of information security (Figure 1), which are Confidentiality, Integrity, and Availability [C+07].

- **Confidentiality** ensures that only authorized users can access the protected information limiting disclosure and unauthorized use.
- **Integrity** is focused on ensuring the authenticity of the protected information. In particular,

[2]https://www.iso.org/obp/ui/#iso:std:iso-iec:27000:ed-1:v1:en

in any stage (storage, transit, process), the information may have not been altered by unauthorized users.

- **Availability** ensures that the information to be protected is accessible only by authorized users. In this sense, policies such as fault tolerance, redundancy, reliable backup, and prevention of data loss or destruction can be applied.

Figure 1: CIA triad.

These three pillars are also known as the CIA triad and are fundamental for evaluating information security. In fact, the lack of one of these pillars can facilitate cyber-attack deployments. In this sense, many attacks are deployed toward each one of these pillars, for example, Denial-of-Service (DoS) or Distributed DoS (DDoS) are implemented for compromising Availability, whereas Trojan, Cross-Site Scripting, and SQL injection are for the Integrity. Lately, Port Scan, Eavesdropping, Phishing, and Man in the middle attacks are drawn to compromise Confidentiality.

In cybersecurity, it is important to differentiate concepts like *threat, vulnerability* and *risk*. In fact, a *threat* is defined as an incident with the potential to harm a system, i.e, more in general, it represents the possibility of a successful cyber-attack. A *vulnerability* is a weakness in the system that an unauthorized user (hacker) can exploit, whereas a *risk* measures the potential damage that a threat generates when a vulnerability is exploited. These three cybersecurity elements are usually reported with the equation: $threat + vulnerability = risk$.

The National Institute of Standards and Technology[3] (NIST) has created a Cybersecurity Framework[4] (CSF) that consists of three main components: *Core, Tiers,* and *Profile*. Core represents a set of guidelines that should be used by private and public entities to improve cybersecurity infrastructure [She14]; Tiers provide information about how an entity views cybersecurity risks and how it should handle them; and finally, Profile is in charge of the alignment of standards and practices with the Core framework. Although, the NIST CSF is not designed to replace existing processes, it continuously evolves for including new changes in cybersecurity threats and technologies. In this way, it can effectively help private and public entities to evaluate their cyber-risks and develop a proper roadmap of improvements.

[3]https://www.nist.gov/
[4]https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf

For the purpose of this thesis, we are interested in introducing the 5 functions that compose the Core component (Figure 2), which are:

1. **Identify:** allows one to detect which process (asset, system, data, service) is more exposed to cyber risk and needs to be protected;
2. **Protect:** suggests applying appropriate safeguards for protecting the identified process (asset, system, data, service) and ensuring delivery of critical infrastructure services;
3. **Detect:** allows one to develop and implement solutions for quickly detecting cybersecurity incidents and events;
4. **Respond:** allows one to deploy appropriate countermeasures and actions for mitigating the impact of a (detected) incident;
5. **Recover:** proposes to restore the process (asset, system, data, service) affected during the incident for regaining system health (resilience).

Figure 2: NIST Cybersecurity Framework (CSF) Core.
https://www.nist.gov/cyberframework

Once the main cybersecurity concepts are introduced, this thesis is focused on novel solutions that can improve the *Detect* function. In fact, cybersecurity is a rapidly evolving sector, and new knowledge promoted by using new technologies has led to the creation of more "intelligent" attacks even more involved in cybercrimes. This situation has raised the need to create new defense solutions that promptly detect these new threats and at the same time are able to adapt themselves to the continuous changes. In this scenario, Artificial Intelligence (AI) represents a key technology for improving "traditional"detection techniques which in turn can be used for quantifying cyber risks, optimizing cybersecurity operations [SKB+20], increasing the resilience of the infrastructure, supporting Law Enforcement Officers (LEOs) investigation, and so on. In this thesis, our analyses are focused on using two AI approaches, *classification* and *anomaly detection*, to cybersecurity detection tasks.

2.1.1. Classification

Classification is a supervised learning task in which input observations are classified into two or more classes, as shown in Figure 3. In supervised learning, models need to be trained with labeled data in order to perform their tasks. In particular, the number of possible classes (labels) is a finite discrete number, otherwise, if labels are real numbers, we would be talking about a *regression* tasks. According to the label distribution, we can have three kinds of classification: *binary, multi-class* and *multi-label*. As the name suggests, in binary classification, the dataset is composed of samples that belong to two classes. In this sense, a classifier is trained for predicting the class to which a new sample belongs. In multi-class classification, the initial dataset has three or more classes, whereas, in multi-label classification, every single sample can belong to one or more classes.

The observations are usually split into training, validation, and test dataset. The training dataset is used by the classifier to understand how the input data and their classes are related. In this way, it tries to generalize these relations that are relevant for the classification of new and unknown observations. The validation dataset is used for tuning the learning process of the model and for evaluating its performance. In particular, this operation is important for detecting problems such as *underfitting* and *overfitting*, which can affect the quality of the classification. More specifically, underfitting happens when a model is not able to learn the relations between data and classes during the training. Hence, it may neither classify the training data nor new observations correctly. On the other hand, overfitting refers to a model which is not able to generalize the relations learned during the training process, showing very low performance in the validation operation [NMB+18]. In this sense, a good approach is to reach a balance between these two scenarios. Finally, the test dataset is used for recreating a real-world application, i.e., to assess the model performance with new unseen data.

There are two main ways to learn classifiers, one based on lazy learners and the other based on eager learners. In the first case, the models store the information received during the training and perform the classification evaluating how the test data are similar to the stored ones. In this sense, these models show very quick training time but they need more time for the prediction. On the other hand, eager learning uses the training dataset to actually learn the dataset distribution and create a space used for the classification of new samples. This approach generates models which take more time for the initial training but are more quickly in the prediction.

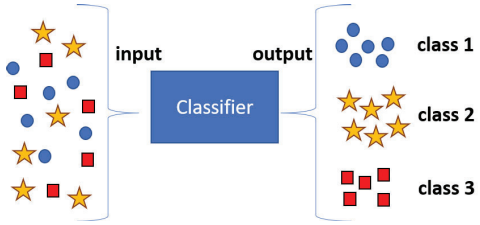


Figure 3: Example of multi-class classification.

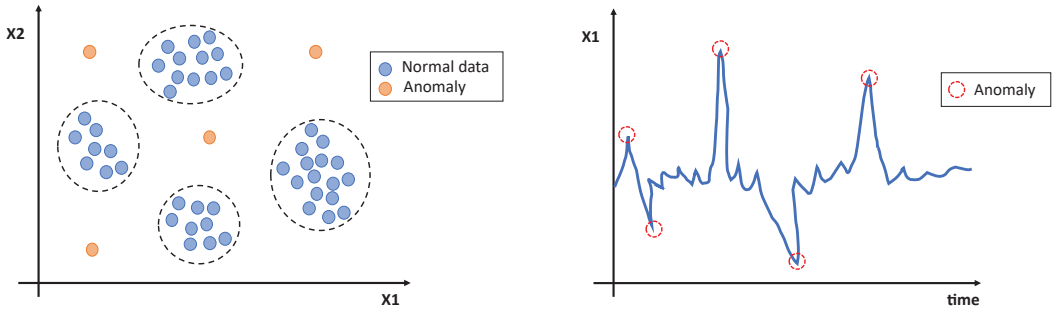
Generally, to obtain a good classification model, it is necessary to train it with a balanced

data set, i.e, a dataset with a homogenous population of all the desired classes. In this way, all classes have equal importance. On the contrary, a class imbalance problem can negatively affect the quality of the model (see Section 2.3). The quality of the classification, as well as limitations introduced by underfitting and overfitting problems, can be evaluated using common metrics such as *Accuracy*, *Precision*, *Sensitivity (or Recall)*, *F1-score*, and *Area Under the Receiver Operating Characteristic Curve (AUC-ROC or AUROC)* [FHOM09, HOFFR12]. In particular, *Accuracy* represents the overall effectiveness of a classifier, however, it can lose reliability when the dataset is not balanced. *Precision* is a measure of a classifier's exactness, and it is used to evaluate the quality of the model in the classification, whereas *Sensitivity* represents a measure of a classifier's completeness, i.e., it is used to evaluate the amount of correct element classified. These two metrics can be combined through a harmonic mean in a unique measure called *F1-score*. The F1-score shows the relation between actual positive labels and those given by the classifier, and it works well on imbalanced data. Finally, *AUC-ROC* is used for visual comparison of classification models and it represents the classifier's ability to distinguish between classes.

2.1.2. Anomaly detection

Anomaly detection (AD) is a task in which models learn the distribution of given data and try to detect points that are different from the norm, thereby classifying them as anomalies [CBK09]. Although AD techniques are sometimes referred to as an unsupervised problem with unlabeled data [ALPA17, LL05], in general, they also include the possibility of considering labeled data for this task [CC19, Agg17]. AD approach is a very relevant task that can help analysts and experts detect potentially dangerous situations and at the same time reduce the amount of information to look at. In fact, AD can be used for both threat detection and threat prevention in a wide variety of domains, such as bank or insurance fraud detection, cybersecurity intrusion detection system (IDS), fault detection in safety-critical systems, and so on. However, it is to be noted that, an anomaly does not represent directly an attack sample or a malicious activity but it represents something noisy, novelty, or an outlier with respect to normal behaviors previously defined. In particular, these behaviors can be defined differently according to the available data, as shown in Figure 4. For example, when a 2-dimensional dataset is used (Figure 4a), points that have similar values for their characteristics generated dense areas which can be used for identifying normal behaviors, whereas points that are sufficiently far away from these areas can be detected as anomalies. Instead, when a time-series dataset is used (Figure 4b), normal behavior can be defined by analyzing all the temporal values, and the points that are far away from it can be labeled as anomalies. In simple cases, this normal behavior can be computed using statistic operations such as the average, percentiles, and the variance.

Initially, static rule-based solutions were implemented for detecting anomalies. However, these solutions are less flexible and can require high efforts, especially in management operations. In fact, in cases in which normal behaviors slightly change periodically, the solutions need constant reviews for removing or adding new static rules [AA18]. In this sense, the evolution of AI has led researchers to implement anomaly detection systems that are more "intelligent" and with greater adaptability [VWK+20]. In fact, several detectors based on AI and ML paradigms are able to directly define normal behaviors and automatically change these definitions over time [AOC07,



(a) Anomalies in a 2-dimensional dataset. (b) Anomalies in time series dataset.

Figure 4: Example of anomalies in different datasets.

ONJ13]. Hence, they allow the detector to evolve with the data and the system, facilitating its deployment in a new environment.

Traditional AD techniques are based on generating an *anomaly score*. This value represents a numeric value for each input instance which is used for defining it as normal behavior or not. In many cases, this decision is taken by setting a threshold that can be decided a priori or dynamically computed during the training process.

According to [CC19,Agg17,GU16], anomaly detection techniques based on ML algorithms can be separated into *supervised AD, semi-supervised AD* and *unsupervised AD*. This classification can be performed according to their training setup and the usage of the labeled data.

- **Supervised AD** represents an approach that recalls the classification task (Section 2.1.1) when a specific class corresponds to anomalies. In this sense, although the problem is addressed in the same way (classification task), conceptually, in AD, this approach shows two relevant problems that affect its usage and the quality of the results [Agg17, GU16]. On the one hand, anomalous instances are usually strongly outnumbered with respect to normal ones, generating highly imbalanced classification problems. On the other hand, especially in cybersecurity, it is not easy to find labeled anomaly datasets to be used in the training phase. For this reason, as also admitted in [Agg17,GU16], this approach is not very relevant for AD tasks.

- **Semi-supervised AD** is implemented following the one-class classification idea, i.e, by using a training dataset in which there is only information of a single class, which represents the normal behavior. In this way, the model should learn to produce a low anomaly score only for instances that belongs to the normal behavior, whereas it should show a high anomaly score for instances that are detected as outliers. Although this approach recalls several concepts of *semi-supervised learning* [VEH20], these two strategies should not be confused. Support Vector Machines (SVMs) [WYA19], Autoencoders [ZP17], and Isolation Forests [XWMZ17] are examples of ML technology that can be used as semi-supervised AD.

- **Unsupervised AD** is implemented using the whole dataset at once, without considering the label information. In this way, the model learns to create areas (or clusters) with similar

data which represent the normal behaviors, whereas all the points that are not clusterized are identified as anomalies. Cluster-based techniques such as Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [TK11], k-Nearest Neighbors [Su11], and Ordering Points To Identify the Clustering Structure (OPTICS) [AMKM17] can be used as unsupervised AD.

2.2. Graph Representation

In this section, several concepts related to graphs and their analysis are introduced. In particular, in Section 2.2.1, graph theory definitions and a taxonomy used in this thesis, are introduced, whereas in Section 2.2.2, common techniques used for extracting graph-based structures are described. Finally, in Section 2.2.3, the learning methods that can be applied directly using graph information are discussed.

2.2.1. Graph Theory

The aim of this study is to exploit graph structures in cybersecurity applications in order to improve classification performance. More specifically, the idea is to translate a time series information into graph-based structures which can represent complex entity/node behaviors highlighting their interactions and changes over time. For this reason, following the definition presented in [Bol13], several key concepts related to graph theory are introduced.

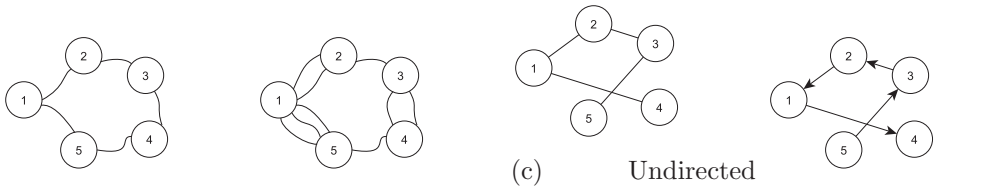
Definition 1. A *graph* G is defined as the ordered pair $G = (V, E)$, where V represents vertex or node set and E is an unordered pair of elements of V called the set of edges. The number of nodes and edges for G will be $|V|$ and $|E|$ respectively.

Definition 2. A graph is called *simple graph* if an edge connects two different vertices and there are no two edges that connect the same pair of vertices (Figure 5a). On the other hand, if the graph has multiple edges that connect the same pair of vertices, it is called *multigraph* (Figure 5b).

Definition 3. An *undirected graph* is a graph in which the edge set is composed of unordered vertex pairs, i.e, edges can be traversed from any direction (Figure 5c). A *directed graph* (or *digraph*) is a graph in which the edge set is composed of ordered vertex (node) pairs, i.e, edges can only be traversed from the specified direction (Figure 5d).

Definition 4. Let $G = (V, E)$ be a graph. Two vertices $u, v \in V$ are *incident* with the edge $e_i \in E$ iff $e_i = \{u, v\}$. A *walk* consists of an alternating sequence of consecutive incident vertices and edges that begins and ends with a vertex. A *path* is a walk without repeated vertices.

Definition 5. Let $G = (V, E)$ be a graph and v_i a node in it ($v_i \in V$). A path that starts and ends in v_i is called *cycle*. A graph with at least one cycle is called *cyclic graph*, whereas a graph with no cycles is called *acyclic graph*. Graphs in Figure 5a and Figure 5c can be also seen as cyclic graph and acyclic graph, respectively.



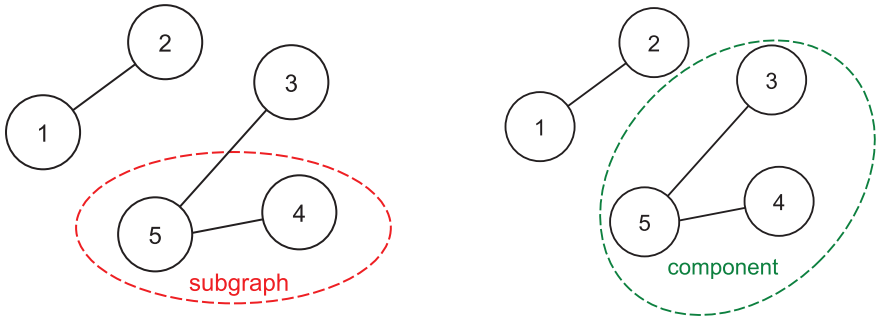
(a) Simple graph. (b) Multigraph. (c) Undirected graph. (d) Directed graph.

Figure 5: Graph structures based on edges.

Definition 6. Let $G = (V, E)$ be a graph and u is a vertex in it. If G is an undirected graph, the *degree* of u is the number of edges incident on it; if G is a directed graph the *indegree* of a node is the number of edges leading into that node and its *outdegree* is the number of edges leading away from it. A graph in which all the nodes have the same degree d is called *regular* or *d-regular*. The graph in Figure 5a can be also seen as a 2-regular graph.

Definition 7. Let $G = (V, E)$ be a graph. G is *connected* iff $\forall u, v \in V$, $\exists$ a sequence of edges $e_1, e_2, ..., e_n \in E$ such that there is a path from u to v. More specifically, there is a path from any node to any other node in the graph. A graph that is not connected is said to be *disconnected*. All the graphs in Figure 5 can be also seen as connected graphs.

Definition 8. Let $G = (V, E)$ be a graph. Then, $G' = (V', E')$ is a *subgraph* of G iff $V' \subseteq V$ and $E' \subseteq E$ (Figure 6a). It can be written as $G' \subseteq G$. Every connected maximal subgraph is called a *component* of the graph (Figure 6b). A graph composed by many *components* represents a disconnected graph.



(a) Example of subgraph. (b) Example of component.

Figure 6: Graph internal structures.

2.2.2. Graph Construction Techniques

The transformation among domains can always cause a distortion or loss of the initial information. These problems can even be more relevant when the aim is to convert vector-based

data or a time series to a graph data representation, since several heuristics need to be defined to correctly describe the problem and to encode as much information as possible [SZ12]. Usually, in cybersecurity, datasets are composed of information gathered from networks and are represented as time series, i.e, each row contains features related to connections, transactions, logs, events between one or more sources and destinations (depending on the granularity of the dataset). For this reason, the conversion of this time series data into a graph-based structure is not trivial, and in many cases depends on the problem to be addressed.

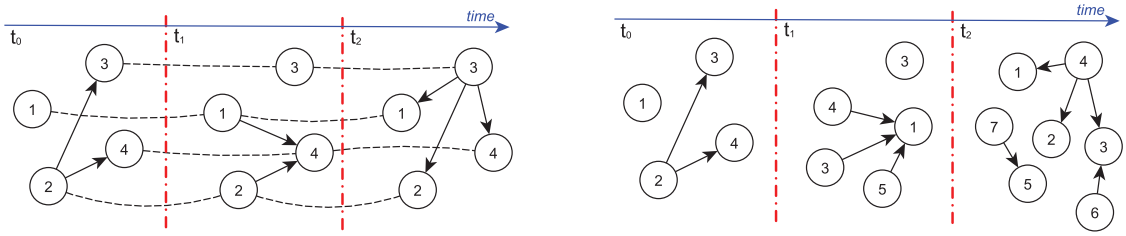
Several techniques used for converting vector-based and time-series data into a graph representation are discussed in [SZ16]. In particular, common approaches are based on creating networks using k-nearest neighbors. These methods propose to consider each row of the dataset as a node of the graph and then they establish links only between the k most similar vertices [SZ12, CZ18]. A similar approach is called ϵ-radius technique in which edges are created if two nodes have a dissimilarity value less than a fixed ϵ [SZ12, CZ18]. These two techniques treat both dense and sparse regions in the same way, for this reason, they are also known as static network methods. On the other hand, techniques such as linear neighborhood [WZ07], b-matching network [SZ16], and clustering heuristics technique [CHZ13] represent adaptive or dynamic methods. The former is based on the assumption that a data point can be linearly reconstructed from its neighborhood, and hence, it proposes to approximate an entire graph by a series of linear neighborhood data points, while linear embedding is used for dynamically computing the edge weights. The b-matching techniques ensure that the nodes in the graph have exactly b neighbors, producing a regular graph. The clustering heuristics technique uses a single-link heuristic for constructing connected and sparse networks. This single-link heuristic ensures to keep the cluster structure of the original dataset.

However, in several real-world applications, edges could connect more than two nodes at a time, different graphs can be interconnected between them, or even they can share a specific node although they represent distinct problems. In this sense, k-uniform hypergraphs (or k-graph) [Ber84] is a technique based on hypergraphs used for representing edges with multiple vertices, whereas multilayer networks [KAB+14] are used for describing the interconnections among distinct interdependent graphs, also called layers. These layers can represent distinct problems such as the bus network, the tramway network, the subway network, and so on. Graph generalization techniques for time-varying networks, i.e, networks that may vary in time, have been proposed in [HS12, FMS13]. On the other hand, MultiAspect Graphs (MAGs) [WFZ16] are introduced for integrating several aspects (a finite list of sets) in the same representation, such as layers and/or time. This representation is mainly used in dynamic complex networks, for example in face-to-face in-person contact networks, mobile phone networks, urban transportation networks, brain networks, and social networks. Time-Varying Graphs (TVGs [CFQS12]) are specific MAGs based on multilayer and time-varying networks. TVGs can be drawn using an algebraic representation [WZF15]) as shown in Figure 7a, in which spatial edges are indicated as solid lines, whereas temporal edges are drawn as dashed lines.

Furthermore, several representations have been introduced for working with specific use cases. This is the case of the Traffic Dispersion Graph (TDG) introduced in [IPF+07] for working directly with network traffic data. These TDGs are graphical representations of various interac-

tions of a group of nodes ("who talks to whom"). The authors exploit network-wide interactions of hosts for extracting graph structures from network traffic datasets, considering each node as a distinct IP address and edges as their communication flows. Furthermore, although the definition of the TDG's nodes is a simple process, the principal task is the definition of the edges. This can be done based on the available information, for example, the first sent packet, the amount of exchanged information, the protocol used, and so on. These edges can be both directed or undirected, according to the final goal. This mapping represents a viable solution when inputs are network traffic data. In particular, it allows monitoring, analyzing, and visualizing the relations among defined nodes using social interaction paradigms [IPF+07].

In [IFM09], the concept of TDG is extended by introducing a temporal factor, i.e.. the authors propose to split the initial data into subintervals, also called temporal batches, and extract a graph from each subinterval. In particular, as shown in Figure 7b, temporal TDGs do not have temporal relations, i.e., the *node 1* in t_0 can represent a different entity from the *node 1* in t_1. A similar approach is presented in [JSZ09], in which authors propose to use Traffic Activity Graphs (TAGs) for linking IP addresses (nodes) with flows that represent communications or interactions (edges). The authors state that TAGs can be used to represent network traffic activities including HTTP, Email, DNS, peer-to-peer (P2P), online chat, and gaming applications. In [DSSVW11], authors propose to use a protocol graph, i.e., a graph that represents Secure Shell Protocol (SSH) information as a directed multigraph relating IP addresses (nodes) through the activated SSH sessions (edges). Each edge is characterized by features like the start and finish times of the flow. Furthermore, the author proposes to decompose this protocol graph into telescoping subgraphs (TSGs) that gather information about a single user or attacker. A TSG represents a direct acyclic graph (DGA) that satisfies predetermined telescoping conditions defined by the authors.



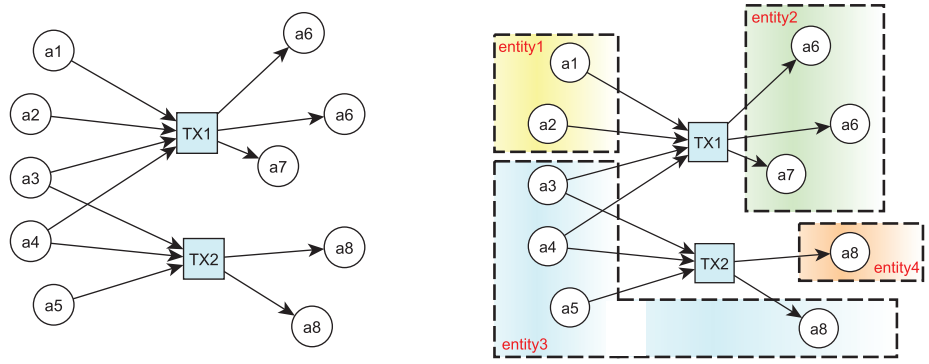
(a) Example of Time-Varying Graphs with algebraic representation.

(b) Example of temporal Traffic Dispersion Graph.

Figure 7: TVGs and TDGs comparison.

Other specific graph representations are related to money transactions since this information has a natural graph structure when connecting two or more banking accounts, crypto wallets, or addresses directly in a transaction. For example, using cryptocurrencies, it is easy to compute an address-transaction graph [FKP15] like the one reported in Figure 8a. This graph is directly obtained by using the information gathered from the blockchain and provides an estimation of the flow of crypto money linking public key addresses over time. The vertices represent the addresses $(a_1, a_2, ..., a_N)$ and the transactions $(tx_1, tx_2, ..., tx_M)$. The directed edges (arrows) between entities

and transactions indicate the incoming relations, while directed edges between transactions and entities correspond to outgoing relations. Each directed edge can also include additional features such as values, time-stamps, etc. Furthermore, starting from the address-transaction graph, and knowing the good practice related to crypto anonymization, it is possible to cluster addresses belonging to the same logical user (or *entities*). An entity is defined as a person or organization that controls or can control multiple public key addresses. This definition allows us to simplify the address-transaction graph into the entity-transaction graph, as shown in Figure 8b.



(a) Address-transaction graph with cryptocurrency data.

(b) Entity-transaction graph with cryptocurrency data.

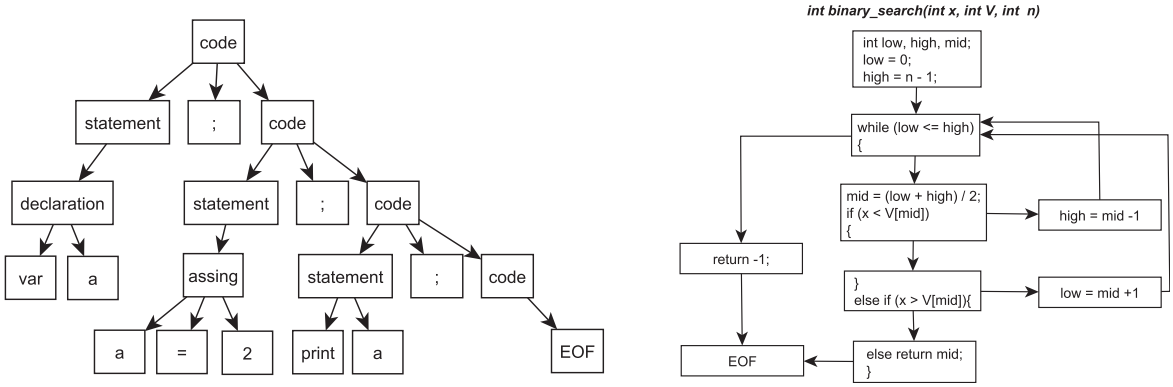
Figure 8: Graph representation for money transactions.

Abstract Syntax Tree (AST) and Control Flow Graph (CFG) are two graph representations used with program code data [PRP21]. AST allows a tree representation of the structural and content-related information of a program code (Figure 9a). On the other hand, CFG is a graph representation that models all of the paths of execution that a program might take during its lifetime. In the graph, the vertices are the basic blocks, sequential code without branches or jump targets, of the program, and the edges represent the jumps in the control flow of the program, as shown in Figure 9b. One of the advantages of this representation is that it has been shown to be very difficult for a polymorphic virus to create a semantically similar version of itself while modifying its control flow graph enough to avoid detection.

In this thesis, the method used for creating the initial graph is performed according to the input information and the available knowledge of the domain. In this sense, a direct representation of address-transaction is used when Bitcoin blockchain data are used, whereas temporal TDGs are used for representing the network traffic dataset, following the indication provided in [IFM09]. Finally, CFGs are used for extracting graph structure from source codes.

2.2.3. Graph Machine Learning

ML has gained massive success in the last years, being used in tasks related to the classification of both structured and non-structured data such as images, videos, speech, etc. In this scenario,



(a) Example of AST. (b) Example of CFG.

Figure 9: Graph representation for program code data.

problems characterized by non-Euclidean spaces that include complex relationships and interdependence between samples have increasingly emerged. This is the case of graph data, in which these non-regularities in the structure have led researchers to create new learning paradigms that work directly with the graph information for performing the classification task.

Graph learning techniques can be categorized depending on their applications, i.e., if they are used for performing graph, node, community, or link classification/regression [Ham20], or according to the ML algorithms involved, i.e, if they use graph signal processing (GSP), matrix factorization, random walk, or DL methods [XSY+21]. Among these techniques, the promising results obtained by graph learning techniques based on DL methods have attracted researchers and the community to better explore their possible applications, converting them into powerful solutions for graph analysis [ZCZ20]. However, there are many graph variants that have led to the creation of different graph DL architectures. In particular, Wu et al. [WPC+20], separate graph DL techniques into four different categories:

1. *Recurrent GNN (RecGNN):* these algorithms use recurrent neural architectures. More specifically, they are able to generate outputs by analyzing the state embedding of a node which also contains the information about the neighborhood of the node itself [ZCZ+18,SGT+08].

2. *Convolutional GNN (convGNN or GCN):* these techniques generalize the operation of convolution directly using the graph representation. The main idea of these approaches is to use high-level node representations and aggregate them through a convolutional operation, as happens in other domains such as image processing.

3. *Graph autoencoders (GAEs):* includes algorithms that recall the concepts of the autoencoders. In particular, GAE techniques are used for encoding the nodes/graphs structures and then reconstructing them by learning the network embeddings.

4. *Spatio-temporal GNNs (STGNNs):* these techniques combine spatial and temporal information. They are able to capture the dynamism of graphs and usually combine the GCN and

RecGNN approaches.

Due to their potential and promising results [JDW+21, YHZ+21], GCNs represent the most interesting and prominent graph DL technology [ZTXM19]. For this reason, in this thesis, GCNs are thoroughly analyzed and adapted for working in cybersecurity applications.

2.2.4. Graph Convolutional Network (GCN)

These networks were introduced in [KW17] for learning the local and global structural patterns of a graph. The aim of GCNs is to consider the relationships among the nodes through a convolution operation resembling the Convolutional Neural Networks (CNNs) applied to 2D or 3D information. In fact, these two technologies (CNNs and GCNs) are very similar in their key concepts. Nevertheless, the first one is built specifically to operate with Euclidean data, whereas the second one is suited for non-Euclidean data, such as the graphs that contain nodes, connections, relations, and unordered information. According to the chosen filter and its application, convolution operations can be separated into spatial-based and spectral-based convolution. In the first category, the operation is applied directly on the graph nodes, for example, using a weighted average function over a node and its neighborhood [HYL17, MBM+17]. In the second case, the filters are defined following graph signal processing concepts. In particular, the Laplacian Matrix of the graph is used to perform a Fourier transformation, and the graph filtering operators within [KW17, DBV16]. While it is straightforward to compute the convolution in the spatial domain, it is more complex in the spectral domain. In fact, the convolution in the spatial domain is similar to the "traditional" image convolution, in which the central pixel (now central node) is convolved with its neighbors for determining the new value. This concept is also similar to the message passing techniques used by several RecGNNs methods, such as the gated GNN [LTBZ15]. On the other hand, in the spectral domain, the convolution operation is defined using a scalar vector (a scalar for every node in the graph), the matrix of eigenvectors of the Laplacian Matrix of the graph, and the filter in the Fourier domain. Nevertheless, solving this equation can be computationally complex and unreachable, particularly for large graphs. For this reason, Chebyshev polynomials with k degrees, i.e, a polynomial function that can be computed recursively, can be used for simplifying the equation [DBV16]. Furthermore, Kipf et al. [KW17] demonstrate that a good approximation can be reached by truncating the Chebichev polynomial to get a linear polynomial ($k = 1$) and by performing a renormalization trick to avoid numerical instabilities and vanishing gradients.

GCNs are used in a wide range of applications, especially for detecting similarity among networks and for discovering patterns among the nodes' relations. In [LWK+20], the authors present a GCN-based framework for predicting Microbe-Drugs associations in SARS-CoV-2 and two antimicrobial drugs, whereas Coley et al. [CJR+19] use the GCN to predict the result of a chemical reaction given the components. Moreover, a graph convolutional solution is combined with reinforcement learning to generate molecules in [YLY+18], while a GCN-based framework able to predict the properties of a material by analyzing its crystal structure is presented in [XG18]. GCNs found applications also in social science for implementing recommendation systems [WSH+18, JGH+20]. In the last years, the good results and the potential of this DL graph technology have led resear-

chers to explore their application in other domains like cybersecurity. Zhao et al. [ZLY+20] use the GCN to transform the botnet detection problem into a semi-supervised classification problem, whereas, in a second research [ZYL+20], they propose a new framework for cyber threat discovery based on the processing Indicator of Compromises (IOCs). Oba et al. [OT20] present a solution based on GCN able to analyze a multigraph based on triplets of client IP, server IP and TCP/UDP ports. In [GCZ21], heterogeneous graphs and GCNs are combined for classifying Android malware, meanwhile, for the same task, in [PYT20], the GCN and recurrent networks are used for identifying and learning semantic and sequential patterns. In [YYJ19], the authors disassemble the malware codes in control flow graphs (CFGs) and use GCN for the malware classification task. GCN can be also applied for implementing malicious domain detectors [SYWL20] or for creating an anomaly detection system for threat and fraud detection [JCG+19]. In [ZLL+19] an anomalous edge detection framework based on GCN with an attention model (Gated Recurrent Unit) is presented, whereas in [WSH+20] graph-based information is combined with flow-based data for detecting botnets.

Following these promising trends, in this thesis, we propose to analyze the benefits and limitations of applying graph convolutional approaches directly for entity behavior classification. More specifically, we propose to explore how this technology works with both imbalanced and balanced graphs, and at the same time, we propose to evaluate how the graph relations affect the learning task. In particular, we propose to compare results obtained by GCN with the ones obtained by using other models that do not consider the graph relations in the classification task.

2.3. Class imbalance problem

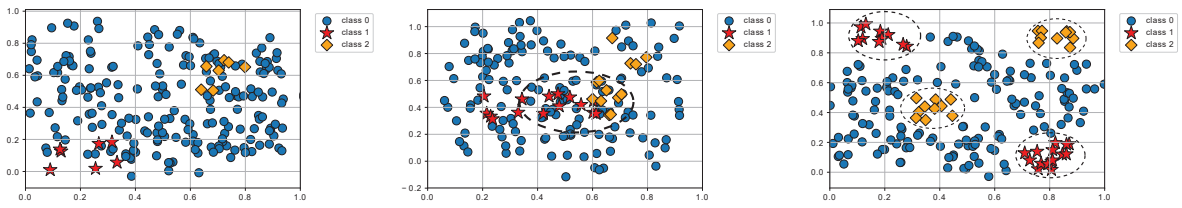
Many real-world applications are characterized by the *class imbalance problem*, i.e, skewed scenarios in which a particular class, category, or event is more frequent than other rare ones. This situation is even more relevant in the cybersecurity domain, where data from network traffic [MS15], fraud transactions [WLC+13], or malware binaries [ODY+19] are analyzed for detecting attacks or illicit operations. In fact, in these applications, it is easier to find information about "normal" events (majority class) than information of rare ones (minority class).

The imbalance problem can be directly exploited for performing an anomaly detection (AD) analysis. This usually happens when the number of the minority observations is very low with respect to the overall observations (high imbalance ratio). In this case, as introduced in Section 2.1.2, unsupervised ML techniques can be used for learning the input data distribution, defining a common trend (or behavior) in the given data, and finally detecting those points (anomalies) that are different from the defined common trend [CBK09]. Of course, in this way, an anomaly is not necessarily related to a threat or an attack, but it represents something rare in the input data distribution. In cybersecurity, this approach is widely used for implementing Intrusion Detection Systems (IDS) [KGVK19].

When the number of observations is enough to describe the minority population, it is possible to directly address the imbalance problem and apply classification approaches, as introduced in Section 2.1.1. If the class imbalance is not addressed, the skewed dataset may introduce a bias that affects the quality of the classification, since the trained models tend to favor the majority class

over the others, making hard to discover robust patterns for under-represented classes [GYD+08]. The main characteristics of a skewed distribution that can degrade the learning performances are related to *small sample size, overlapping,* and *small disjuncts*.

- *Small sample size*: it occurs when the ratio between majority and minority classes is very high meaning that the dataset is affected by a strong class imbalance (Figure 10a). Hence, the amount of information in the minority classes is not enough for generalizing their distribution leading to fail the classification [WC09].
- *Overlapping*: it is present when the samples of the majority and minority classes are mixed in the same space creating ambiguous regions, as shown in Figure 10b. These ambiguous regions are areas in which the boundaries of the classes are not clearly defined and samples of two or more classes can be found [GC17]. This scenario increases the difficulty for the classifier to separate the different samples thus producing misclassifications. Although ambiguous regions can be formed also in balanced datasets, they are very critical in imbalanced datasets since the small number of samples in the minority classes can facilitate their creation [LFG+13].
- *Small disjuncts*: this phenomenon occurs when the minority classes are formed of smaller subclusters of samples, as shown in Figure 10c. This situation increases the complexity of the problem, especially when dealing with imbalanced datasets, since it is difficult to separate the different samples and at the same time to determine if the clusters represent noisy or relevant information [LFG+13].



(a) Small samples size (b) Overlapping (c) Small disjuncts

Figure 10: Example of the main problems in class imbalanced dataset

These phenomena can be mitigated by applying techniques for addressing class imbalance. These techniques are widely used with tabular data and can be grouped into four categories [FGG+18b]:

Cost-sensitive learning. These algorithms use cost values associated with features or associated with classes for driving the training process when imbalanced data are used [ZZ20]. For the case of class imbalance, different costs are usually assigned to the missclassification of the different classes, giving more priority to examples from the minority classes. In this scenario, the crucial task is to define the cost matrix, since a wrong initialization can impair the learning

process. MetaCost, cost-sensitive decision trees, and cost-sensitive SVMs [FGG+18a] are just an example of the most used algorithms that belong to this category.

Data-level preprocessing. These techniques operate directly over the dataset for addressing the class imbalance through a resampling approach. In this sense, according to the process used for obtaining a more balanced population, algorithms can be grouped into undersampling, oversampling, or hybrid methods. Undersampling methods remove samples from the majority class, as proposed by Random Under Sampling (RUS) or Tomek's links (TL [PCSJ20]) algorithms. On the other hand, oversampling strategies are based on adding new elements to the minority class. Well-known oversampling strategies are Random Over Sampling (ROS), Synthetic Minority Over-sampling Technique (SMOTE [FGHC18]), and Adaptive Synthetic (ADASYN [VNVL21]). Finally, hybrid approaches combine both techniques, e.g., for the SMOTE+TL strategy [RAC+20].

Algorithm-level approaches. These approaches address the imbalance problem by directly modifying the learning process. In particular, it is important to analyze which procedures and mechanisms are the most prone to introduce a bias towards the majority class [FGG+18a]. For this reason, these approaches require a deep knowledge of the classifier and the factors that lead to misclassification. However, each model requires its own adaptation, for example, for in SVMs, the kernel can be adapted including a weighting scheme, whereas in Bayesian classifiers it is possible to manipulate class prior probabilities.

Ensemble learning approaches. These techniques are designed to combine ensemble learning models with the previously introduced imbalance techniques [GFB+11]. In fact, although the ensemble models are able to optimize ML results, they do not directly address the imbalance problem. In this sense, the most common approach is to combine ensemble models with data-level preprocessing techniques, i.e., using these data-level preprocessing before training each classifier of the ensemble [FHR18,SKPM20]. On the other hand, other studies propose to combine ensemble learning with cost-sensitive approaches [SKWW07].

However, the application of these techniques when dealing with graph-structured data is not straightforward and can be misleading. In fact, these techniques are focused on the feature space and if applied without modifications to the graph, they can change its internal structure (topology) altering the problem representation. For this reason, it is important to introduce graph sampling techniques that can be used directly over the graph with the aim to address the class imbalance problem without altering the graph topology.

The most common graph sampling techniques are designed for extracting reduced graphs that are "representative" and "look similar to" the original graph, preserving its properties [HL13]. Thus, the aim of these techniques is not directly to address the graph imbalance problem, but to alleviate scalability issues related to large graphs [HL13], to gather information about hidden populations [CWH18], to handle with graph sparsification [SS11], to perform dimensionality reduction and visualization [Raf05], etc. These techniques have been mainly used in statistics, data mining, visualization, and social media analysis. In particular, they can be grouped into three categories [WCA+16]:

- *Node-based samplings*: these techniques are based on extracting a subgraph from the original graph by randomly selecting a subset of nodes. At the same time, all the edges in which the

selected nodes are involved, are selected as well. This is how Random Node [LF06] strategy works, whereas Random Degree Node [SWM05] uses node degree as a probability to select each node.

- *Edge-based samplings*: similar to the node-based techniques, these methods are based on randomly selecting edges in the graphs and the nodes connected to them, as happens in Random Edge techniques. An alternative is represented by DropEdge [RHXH19], which proposes to randomly remove a certain number of edges from the input graph, whereas GAUG [ZLN+20] proposes to use a neural network as edge predictors for data augmentation operation.
- *Transversal-based samplings*: these strategies are based on selecting a node (or edge) in a graph and then expanding it by exploring all the edges/nodes in the neighborhood. For this reason, they are also called "topology-based sampling" or "sampling by exploration". In fact, with the exploration of the neighborhood these techniques are able to preserve the graph topology. Random Walk, Snowball Sampling [SKR+16] and Forest Fire Sampling (FFS) [LF06] are among the most common transversal-based samplings techniques. However, these approaches can introduce a bias toward high-degree nodes, i.e, during the exploration, they favor high-degree nodes more than low-degree or peripheral ones. Moreover, if the graph is composed of many components, these methods could get stuck in a single component during the exploration, leaving part of the graph structure unexplored.

Although these techniques can be directly used over graph structures, they are designed to perform a graph sampling to address scalability issues when analyzing large graphs, and not for addressing the class imbalance problem. In fact, as introduced in [HL13, WCA+16], node-based samplings and edge-based samplings do not respect the graph topology and therefore, they can alter the problem representation. On the other hand, transversal-based sampling cannot be directly used with disconnected graphs. These techniques are more proned to introduce a bias towards high-degree nodes, i.e, they favor the selection of high-degree nodes removing mainly the low-degree or peripheral nodes. For this reason, as part of this thesis, we propose novel graph sampling techniques which can work directly on disconnected graphs avoiding component topology changes for addressing the graph imbalance problem.

2.4. Generative Adversarial Networks

Generative Adversarial Networks, or GANs, are deep-learning-based generative models introduced in [GPAM+14], whose aim is to learn a data distribution and generate realistic synthetic samples which follow that distribution. A GAN is composed of two neural networks: a Generator (G) and a Discriminator (D). These networks compete with each other, thereby increasing their ability to learn from each other. In particular, the aim of G is to generate synthetic samples from a random uniform distribution, while D evaluates their authenticity, i.e., it determines if samples belong to the synthetic or real distribution.

In this adversarial learning, the generator output is connected directly to the discriminator input, whereas the discriminator output is connected with the generator through the backpro-

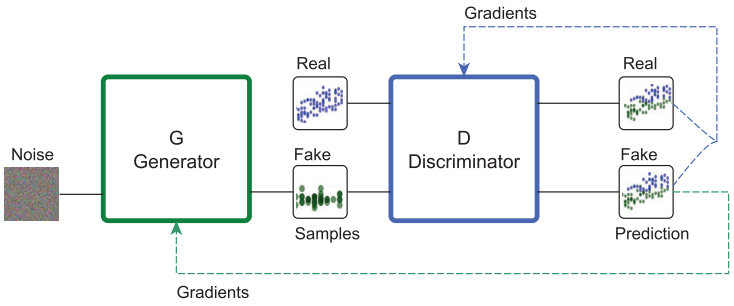


Figure 11: Generative Adversarial Network

pagation of the gradients, as shown in Figure 11. In this way, their loss functions are involved in a zero-sum non-cooperative game, in which one network tries to maximize the effects of its actions, while the second network tries to minimize its effects. This task, also called *minimax* problem, can be mathematically formulated using Equation I.1 [GPAM+14]. This equation represents the two-players functions as $V(D, G)$, while z and x represent a random uniform sample and a real sample from the input distribution, respectively. $G(z)$ represents the generator's output, i.e., the synthetic data, $D(x)$ is the discriminator's estimation of the probability that x is real, while $D(G(z))$ is the discriminator's estimation of the probability that $G(z)$ is real. $E_{x \sim p_{data}(x)}$ is the expected value over all real data associated to the probability distribution of the real data $p_{data}(x)$ and $E_{z \sim p_z(z)}$ is the expected value over all random inputs to the generator associated to a predefined prior noise distribution $p_z(z)$.

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)}[\log(D(x))] + E_{z \sim p_z(z)}[\log(1 - D(G(z))] \tag{I.1}$$

Following Equation I.1, the GAN can be seen as a dynamic learning system whose training goal is to find an equilibrium between the two players. In this sense, training converges when one player will not change its actions regardless of what the opponent may do [O+04], reaching the well-known Nash equilibrium (NE) [FRL+17]. This equilibrium in the cost function can be computed using stochastic gradient descent (SGD). More specifically, SGD [GPAM+14] represents a common optimization technique that computes the gradient and updates for both G and D simultaneously [Rud16]. Nevertheless, other studies [RLWFM17, MNG17] propose alternative optimization algorithms since, in several scenarios, SGD is not able to converge properly.

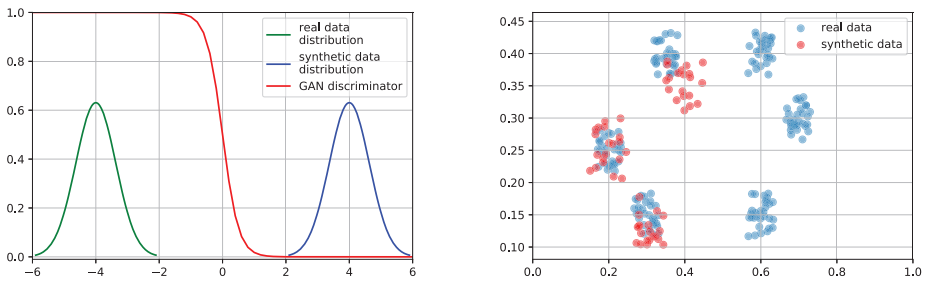
Despite its power, due to the inappropriate design of network architecture, the cost function, hyper-parameters, and variety of optimization algorithms, common collateral effects [LMPS17] can affect the GAN performance. The major drawbacks that can generate limitations in the GAN training and generation of synthetic samples are mainly *non-convergence, vanishing gradients* and *mode collapse*.

- *Non-convergence*: it occurs when it is not possible to find a NE between G and D. This commonly occurs in GD when the optimization gets stuck in local minima or saddle points.

In this scenario, the two players reverse each other's progress generating a training stalemate that can produce a solution that does not converge, or even oscillation in the model parameters and thus instability [Goo16]. To alleviate this issue, in [AB17] and [RLNH17], the authors proposed methods for alleviating the non-convergence situation: in the first case, they added noise to D's inputs, whereas in the second they penalized D's weights.

- *Vanishing gradient*: it occurs when D is so strong that it overpowers G, i.e, it is able to detect fake and real samples without problems limiting the information backpropagated to the generato, as shown in Figure 12a. This phenomenon is common in DL models that use gradient-based optimization as well as in the traditional GAN [GPAM$^+$14] implementation. In particular, during the GAN training when $D(x) = 1$, $\forall x \in p_{data}(x)$ and $D(G(z)) = 0$, $\forall G(z) \in p_z(z)$ the gradient is squeezed close to zero and thus no useful information is propagated to the generator (flat areas in Figure 12a). This effect generates slows down training which may even lead to non-convergence. Solutions for addressing the vanishing gradient are mainly based on using alternative loss functions such as those based on least-squares [MLX$^+$17], Wasserstain distance [AB17], or f-divergences [NCT16].

- *Mode Collapse*: also known as the Helvetica scenario, it occurs when G learns just a small part of the input distribution that does not represent the entire population. Therefore, G generate samples with low diversity, as shown in Figure 12b. In the worst case, the model "collapses", generating always the same sample. This situation can easily happen when D does not have high generalization capabilities, for example, when it gets stuck in local minima and hence, G learns how to "fool" it without exploring the entire input distribution. This scenario represents a very critical and relevant problem in GAN implementations, since its main goal is to create a variety of realistic samples. Although it is easy to detect and visualize the mode collapse especially when images are used, its solution is not trivial. In [MPPSD17], the authors propose to update the G function not only considering the current backpropagation of D, but also future outputs in order to avoid over-optimization.



(a) Vanishing gradient (b) Mode collapse

Figure 12: GAN drawbacks

In order to mitigate these effects, different GAN architectures have been designed and implemented. For this reason, apart from the traditional GAN architecture, also called *Vanilla GAN*,

many studies [WSW21, SC21] are mainly focused on the re-engineering of the network architectures, looking for new cost functions, or proposing alternative optimization algorithms. In fact, it is to be noted that, GAN architecture and its configuration needs to be changed according to the problem to be addressed, the application domain, and of course used dataset.

The use of GANs appears to be promising as numerous studies have demonstrated good results in the image and speech processing domain. The main goal of these adversarial networks concerns the analysis and generation of synthetic images - for example, creating super-resolution images [SZW+19], realistic photographs of human faces [HZS+20] or image inpainting [XZLH20]. For the speech domain, GANs are used to enhance the quality of speech contaminated by additive noise [PBS17], to improve Neural Machine Translation (NMT) results by generating human-like translations [YCWX18] and for emotion recognition by creating synthetic audios from audio-visual datasets [AHA20]. In the context of (supervised) ML problems, GAN data augmentation has generated clear improvements in classifier performance [OHB19, DB18], especially in (bio-)medical research, where analyses suffer from class imbalance due to privacy issues and the rarity of some pathological findings [MMKSM18].

Only a few recent studies apply GANs in the context of cybersecurity analysis for generating new cyber-attack samples from existing data [MSS+19, YM19], to address the imbalance problem in encrypted traffic datasets [WLY+20] or as anomaly detectors for potential security attacks on mobile phones [SKK20]. Two interesting approaches are introduced in [ZBB+20] and [HWH20]. In both cases the authors try to use GANs to address a cybersecurity class imbalance problem, however, they use very limited datasets and the obtained results were strongly affected by GANs downside effects (mainly mode collapse). Therefore, no clear classification improvements were produced. For this reason, inspired by the approaches presented in [ZBB+20, HWH20], as a part of this thesis, we investigate how GANs models should be adapted for working with cybersecurity tabular data with the aim to address the class imbalance and improve the final behavior classification. More specifically, since this approach is based on creating synthetic samples for the minority classes, it can be seen as an alternative to the common data-level preprocessing techniques.

2.5. Cyber Threat Intelligence

Although there is no a single definition for Cyber Threat Intelligence (CTI), according to Gartner research [McM13], *"CTI is the task of gathering evidence-based knowledge such as context, mechanisms, indicators, implications, and actionable advice, about an existing or emerging menace or hazard to assets that can be used to inform decisions regarding the subject's response to that menace or hazard"*. On the other hand, based on Dalziel [Dal14], the CTI information should be *relevant*, i.e., the information should be related to the organization's interests; *actionable*, i.e., it should be clear for defining prompt actions/responses; and *valuable*, i.e., it should contribute to any useful outcome. Following these definitions, we can state that CTI allows experts and practitioners to unveil and understand attacker motivations, targets, and actions. This novel-generated knowledge can be used on the one hand, for creating and applying new ad-hoc countermeasures able to adapt themselves to attacker behavioral changes. On the other hand, this new knowledge can be used for training the defenders, increasing their expertise.

Several studies try to define a unique CTI ontology such as Targeted Attack Premonition using Integrated Operational data (TAPIO) [SW15], Situation and Threat Understanding by Correlating Contextual Observations (STUCCO) [IBN+15], a three-level modular analysis for cyber security called CRATELO [OCWM14], Cyber-investigation Analysis Standard Expression (CASE)[5], and security ontology [FE09]. However, as presented in [MB17], it is not possible to define a standardization in terms of taxonomy, threat actor motivations, goals, and types, since all these ontologies are built for specific applications and they are not prone to interoperability. On the contrary, in terms of sharing threat information, Structured Threat Information eXpression (STIX) [Bar12] represents one of the most used approaches [SSMB17].

Following military concepts and definitions, CTI can be separated into four sub-domains [CR15, TR18]: *Strategic, Operational, Tactical* and *Technical threat intelligence*. The first sub-domain contains high-level information that can be used for identifying and evaluating unknown threats and risks. In this case, usually, reports, briefings, or conversations are held for defining policies and strategies. The second sub-domain contains information about upcoming attacks. This information is very hard to gather especially for private entities, whereas, it is easier for military and government organizations. The third sub-domain contains information about how attackers deploy their offensive, for example, which tools or methodologies they use. In this sense, technical or white papers are used to gather attackers' information. Finally, the fourth sub-domain contains information about technical resources (firewalls, mail filtering devices, logs systems, etc.), and it shows this information through monitoring tools, dashboards, or analytics tools. This information is extracted from the attacks in the form of indicators of compromise (IOC), which are directly involved in intelligence analysis.

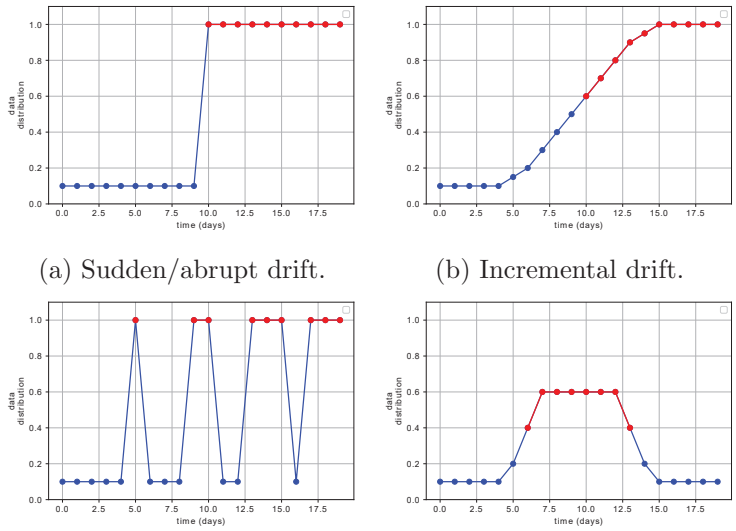
In this thesis, we focus our investigation on technical threat intelligence. Our idea is to extract additional insight from attacker behaviors and how they change over time. In this sense, this information can be seen as an IOC. More specifically, we are interested in analyzing uncontrolled changes in the input distribution that can generate inconsistent and misleading results, especially when ML models are in the loop. In fact, in this case, models trained on old data can show skewed results when novel information is used as input [Tsy04]. This problem is usually known as *concept drift* [WK96] or *dataset shifting* [QCSSL08]. An effective learner (*adaptive learner*) should be able to react to this problem, by detecting such changes and re-adjusting its predictions.

2.5.1. Concept Drift

As previously mentioned, concept drift is generated by uncontrolled changes in the input distribution, and, according to the temporal patterns of these changes, four main different drifts can be detected [LLD+18]: *abrupt, incremental, gradual* or *reoccuring*, as shown in Figure 13. Concept drift is defined as abrupt when the change occurs instantaneously, for example, a sensor that breaks down (Figure 13a). If performance changes are slow in time and in values, they are considered incremental drifts. This is the case, for example, for a worn-out sensor (Figure 13b). On the other hand, if the changes are only slow in time but not in value (Figure 13c), they are considered as gradual drift. Finally, if performance changes are repeated over a certain interval

[5] https://caseontology.org/

(e.g., seasonal), there may be reoccurring drifts, as in an external temperature sensor (Figure 13d). It is to be noted that, although outliers and noise can be seen as points that instantaneously change the data distribution, formally they are not considered as concept drift [LLD+18].



(a) Sudden/abrupt drift. (b) Incremental drift.

(c) Gradual drift. (d) Reoccurring drift.

Figure 13: Drift patterns of changes over time (one-dimensional data).

Concept drift is a highly relevant problem in cybersecurity applications, especially when ML models are implemented. It has been predominantly studied in applications such as anomaly detection [GBR+17], fraud [DPBC+15] and spam [SYZ+16] detection. However, in recent years, concept drift has been started to be investigated also in malware detection. This is because malware continuously evolves over time, since authors try to create new variants for evading their detection. In this sense, a machine learning model can be outdated just after a couple of months. More specifically, in [HMZ+17] and [DGAH+21], authors try to improve classifier's performance by detecting concept drift online, i.e., detecting drift while being in operation and addressing it by retraining the model with more recent data. They propose a rolling-window approach for selecting data to be used to evaluate the performance and eventually retrain the models. Other studies, such as [JSD+17] and [SWL12], propose novel methods for detecting concept drift, however, they do not deeply analyze the relationship between these performance decays and the actual input of the malware classifier.

Inspired by these studies, in this thesis, we propose to analyze concept drift in-depth to improve our CTI analysis. In this sense, our objective is not only to detect concept drift points (where model performance drops), but also to relate this phenomenon with changes in the input features that may have generated them. Through such an approach, drifts could be detected before their actual appearance, and cyber analysts and operators could apply precautionary actions to avoid model failure.

3. Motivation

Once the main concepts on which this manuscript is focused have been introduced, the open problems that motivate this thesis are presented.

- **M1 - Temporal behavior definition.** Usually, in cybersecurity applications, researchers try to directly classify single connections for identifying anomalous or malicious traffic. This task is performed considering network traffic, events, logs, transactions, etc., as times series [BGCML21, CMF19]. In these cases, the idea is to promptly detect the anomalous situations and take the appropriate actions on the connection to avoid more serious damages, increasing network cybersecurity. However, this approach can be very resource-consuming due to the huge amount of information to be processed. Moreover, an incorrect classification of a single connection can lead to making inappropriate decisions, for example, excluding or shutting down a certain entity. In fact, malformed communications carried out by normal entities can result in falsely detected malicious flows (false positives), or malicious activities can be obfuscated in a complex routine to be unnoticed (false negatives) [CKBR06]. For this reason, it is very important not only to evaluate the single connections but also the behavior of the entities that have generated them. This operation can be performed by representing the information as a graph structure defining entities, relations, and topology. Entity behaviors can be defined for each node by evaluating their interactions within the network. However, the graph topology, as well as the behavior definition, strongly depends on the temporal component, since networks and entities evolve over time, and the usage of a static graph cannot properly represent the problem. For this reason, an operation of *temporal dissection* should be used. This *temporal dissection* is based on dividing time into chunks in which the graphs are built. To carry out the temporal dissection, the *temporal resolution* must be set. However, the main problem is that there is no way to know in advance which *temporal resolution* is the best. For this reason, it is very important to study how the *temporal resolution* affects the graph representation, and which values should be used for highlighting changes in the entity behaviors.
- **M2 – Graph imbalance problem.** As already discussed, temporal graphs can be used for highlighting entity behaviors and relations within the network. However, these temporal graphs can be characterized by an imbalanced population, i.e., one or more classes being more represented than others. This imbalance problem is a relevant and common problem in ML applications, especially in supervised learning [FGG+18b]. Furthermore, the imbalance problem can be very challenging in domains where it is hard to detect and collect new observations, as in the case of cybersecurity applications. Clearly, the presence of this phenomenon changes the classification results introducing a bias in favor of the majority classes. The problem is that common data-level preprocessing strategies cannot be directly applied to graph representations. In fact, relational information (topology) must be taken into account for resampling. These techniques are focused on the feature space and if applied without modifications to the graph, the topology would be modified, changing its intrinsic information and altering the problem. In this sense, graph sampling techniques that work directly over the graph can be used. These techniques are based on nodes/edge

selections or in topology exploration [HL13]. In the former, nodes or edges are randomly selected to reduce the representation of the majority class in the original graph, altering the graph topology. In the latter, a random node is first selected and its neighborhood is explored to preserve network topology. However, these techniques showed a bias towards preserving high-degree nodes, i.e, these nodes have more probability to be explored and used in the balanced graph, whereas low-degree and peripheral nodes are the most prone to be removed. At the same time, these techniques show high limitations when disconnected graphs are considered since the exploration process can be trapped in dense components, i.e, components with high-degree nodes, limiting the search. Moreover, components that do not contain the starting node will never be sampled [WCA+16]. For this reason, novel data-level preprocessing methods for addressing graph imbalance problems directly over disconnected graphs avoiding topology changes, need to be introduced.

- **M3 – Cybersecurity imbalance problem.** As previously discussed, the class imbalance is a very common and relevant problem that affects the quality of the classification performance. Although the class imbalance can be directly addressed over the graph, in several cybersecurity applications, even if graph structures are used for representing the information, the classification task is performed using feature vectors (or embeddings) directly extracted from the graphs. In these cases, graph relations are not directly used in the classification phase, and for this reason, the imbalance problem can be addressed by applying traditional data-level strategies. However, they also have limitations, for example, random samplings such as RUS and ROS can easily generate overfitting or underfitting effects, whereas techniques based on neighborhoods such as SMOTE and ADASYN generate new synthetic samples around the original points without considering the overall distribution [GS17]. For this reason, they can introduce noisy information when highly variable datasets are used. In this situation, a novel approach based on generative adversarial learning could have a strong impact. More specifically, generative adversarial networks (GANs) have shown their ability to learn the underlying data distribution from a limited number of available samples, allowing them to be used to address the class imbalance problem. In fact, they have been successfully used in image and video processing for this purpose [BDS18, VPT16]. Nevertheless, their adaptation to cybersecurity applications and their usage with tabular data has not been yet well explored. In fact, the high variability of the tabular data and the difficulty in validating their results compared to the images or videos (which can be visually validated), make cybersecurity GANs implementation very challenging. Moreover, since it is impossible to find a unique GAN solution that works for every scenario, different GAN architectures are currently available in the state of the art. For this reason, a study about which GAN architecture is suitable to address the class imbalance problem in cybersecurity is required. Furthermore, it is interesting to analyze how the GAN training configuration affects the quality of the generated samples.

  M4 – Cyber Threat Intelligence. In cybersecurity, an important task is the implementation of detection systems able to classify new anomalous patterns and anomalies. In this sense, the majority of cybersecurity studies exploit ML and DL solutions to improve detection performance. However, although these paradigms are able to discover new potentially

dangerous information, the same information can also be used by attackers to improve their knowledge and create new disruptive and undetected attacks. In this scenario, trained ML models can become outdated after just a couple of months without noticing [PPJ+19]. At the same time, the improper usage of ML and DL paradigms can result in over-optimistic results and conclusions [AQP+22]. For this reason, it is important not only to focus research on improving classification performance but also on extracting insightful information about data trends and performance drift as well as their evolution over time. In this way, it will be possible to enhance the cyber threat information and improve the domain knowledge.

4. Objective

The main goal of this thesis is *to improve behavioral cybersecurity analysis using machine learning, exploiting graph structures, temporal dissection, and addressing imbalance problems.* This main objective is divided into four specific goals, each one related to one (or more) of the needs introduced previously in Section 3.

- *OBJ1: To study the influence of the temporal resolution on highlighting micro-dynamics in the entity behavior classification problem.* In real use cases, time-series information could be not enough for describing the entity behavior classification. For this reason, we plan to exploit graph structures for integrating both structured and unstructured data in a representation of entities and their relationships. In this way, it will be possible to appreciate not only the single temporal communication but the whole behavior of these entities. Nevertheless, entity behaviors evolve over time and therefore, a static graph may not be enough to describe all these changes. For this reason, we propose to use a temporal dissection for creating temporal subgraphs and therefore, analyze the influence of the *temporal resolution* on the graph creation and the entity behaviors within. Furthermore, we propose to study how the temporal granularity should be used for highlighting network micro-dynamics and short-term behavioral changes which can be a hint of suspicious activities.

- *OBJ2: To develop novel sampling methods that work with disconnected graphs for addressing imbalanced problems avoiding component topology changes.* Graph imbalance problem is a very common and challenging task and traditional graph sampling techniques that work directly on these structures cannot be used without modifying the graph's intrinsic information or introducing bias. Furthermore, existing techniques have shown to be limited when disconnected graphs are used. For this reason, novel resampling methods for balancing the number of nodes that can be directly applied over disconnected graphs, without altering component topologies, need to be introduced. In particular, we propose to take advantage of the existence of disconnected graphs to detect and replicate the most relevant graph components without changing their topology, while considering traditional data-level strategies for handling the entity behaviors within.

- *OBJ3: To study the usefulness of the generative adversarial networks for addressing the class imbalance problem in cybersecurity applications.* Although traditional data-level pre-

processing techniques have shown to be effective for addressing class imbalance problems, they have also shown downside effects when highly variable datasets are used, as it happens in cybersecurity. For this reason, new techniques that can exploit the overall data distribution for learning highly variable behaviors should be investigated. In this sense, GANs have shown promising results in the image and video domain, however, their extension to tabular data is not trivial. For this reason, we propose to adapt GANs for working with cybersecurity data and exploit their ability in learning and reproducing the input distribution for addressing the class imbalance problem (as an oversampling technique). Furthermore, since it is not possible to find a unique GAN solution that works for every scenario, we propose to study several GAN architectures with several training configurations to detect which is the best option for a cybersecurity application.

- *OBJ4: To analyze temporal data trends and performance drift for enhancing cyber threat analysis.* Temporal dynamics and incoming new data can affect the quality of the predictions compromising the model reliability. This phenomenon makes models get outdated without noticing. In this sense, it is very important to be able to extract more insightful information from the application domain analyzing data trends, learning processes, and performance drifts over time. For this reason, we propose to develop a systematic approach for analyzing how the data quality and their amount affect the learning process. Moreover, in the context of CTI, we propose to study the relations between temporal performance drifts and the input data distribution for detecting possible model limitations, enhancing cyber threat analysis.

5. Discussion

In this Section, for each publication that composes this thesis, a brief description and its main results are reported.

5.1. Bitcoin and cybersecurity: Temporal dissection of blockchain data to unveil changes in entity behavioral patterns

In this work, we propose an approach for highlighting changes in the entity behaviors extracted from a graph-based network. For this reason, we firstly propose to define the entity behaviors as the interactions of each node within the network. Then, we focus the study on analyzing how the *temporal resolution*, i.e., the time interval in which a graph structure is extracted, affects this behavior definition. The temporal resolution is analyzed by introducing an operation known as temporal dissection, dividing the initial dataset into several time-fixed batches. Our hypothesis is that a model trained on entity behaviors of a certain batch would achieve a high classification performance when testing it on another temporal batch only if they share the same behavioral patterns, that is, behaviors are not changed.

The temporal resolution is reversely related to the time step size, i.e., the size of the temporal batches. In fact, a low temporal resolution generates batches with a large time step size, which in

turn favor the appearance of macro-dynamics. These macro-dynamics are not only less related to cybersecurity threats, but they can also skew the learning process of a model. On the other hand, a high temporal resolution produces batches with a small time step size which could not contain enough information for properly detecting the entity behaviors. For this reason, the choice of this temporal resolution is not trivial and, in order to find a trade-off between the two cases, we study how it should be selected for highlighting entity behavioral changes.

This study is performed using Bitcoin blockchain data with the main goal of detecting typical, recurrent entity behavioral patterns and at the same time detecting abrupt changes that can be hints of attacks or illicit activities. In particular, six entity behaviors are considered: *Exchange, Gambling, Mixer, Mining Pool, Marketplace* and *eWallet*. Once the operation of temporal dissection is performed over the blockchain, four graph structures [ZEBU19] are extracted from each batch. These graphs are combined by a cascading ML classifier, i.e., an ensemble that stacks several classifiers. This approach is evaluated using four different temporal resolutions to better understand how this parameter affects the micro-dynamics detection.

The results of the proposed analysis have shown that the temporal resolution affects the considered entity behaviors (or classes) differently. Some classes (Exchange, Gambling, and eWallet) have shown a consistent behavior that does not vary significantly overtime in the past 3 years. In fact, using a high temporal resolution slightly affects the performance of the model for these behaviors. Moreover, the results suggest that it is not necessary to use the full blockchain data to identify those classes. Otherwise, other classes (Market and Mixer) worsen their performance when a high temporal resolution is used. This is probably due to the fact that when high temporal resolution is used, too few samples of these classes remain in each batch, as previously discussed. Finally, the last class (Mining pool) changes dramatically its performance when a high temporal resolution is used, although it has a sufficient population in the small batches. Furthermore, this class shows a high performance variability from batch to batch, highlighting its changes over time. These temporal changes can be a hint of illicit activities, as well as a potential cyber-attack suffered, as also shown in [VTM14, JLG+14].

Based on the results that we have obtained in this study, the following conclusions can be highlighted:

- The proposed approach based on temporal dissection not only helps to reduce problem complexity and computational effort related to the Bitcoin data processing, but also successfully shows the usefulness of adjusting the temporal resolution for highlighting entity behavioral changes over time. In fact, the classification results highlight which entities show a relatively consistent behavior from batch to batch (making them less likely to be related to illicit activities or cyber-attacks) and which entities show strongly varying behavior over time (thus revealing short-term changes in behavior that could suggest suspicious network activities).
- It is not possible to select a unique value for the temporal resolution, since each entity class is affected differently. In fact, the study shows that for several classes, a high temporal resolution generates batches with too few samples for defining a behavior. On the other side, a low temporal resolution promotes the classification of macro-dynamics over micro-

dynamics, hiding the temporal changes. Hence, the temporal resolution should be selected according to the specific target, i.e., the class which is most interesting to be studied. More specifically, in our study, for a specific class (Mining pool), it is possible to appreciate that a higher temporal resolution successfully highlights micro-dynamics, and its variability over time could be a symptom of suspicious activities (or cyber-attacks).

- Finally, the analysis shows that models trained with recent data perform well in classifying past behaviors when a high temporal resolution is used. This suggests that generally, it is more difficult to predict the future and that, the class behaviors are continuously evolving. This implies that earlier models do not capture future behaviors as well, but recent models do seem to contain behavioral elements that can help the definition of past data. The aim of this analysis is not to implement a classifier to be used in real applications, since future data are used for predicting past behaviors, but to exploit CTI concepts for extracting more insights, enhancing domain knowledge.

The introduced approach can be used for improving crime investigation, for example for developing new forensics tools which can assist law enforcement officers (LEOs) in uncovering illegal activities within the Bitcoin network. Furthermore, it can be applied to other cybersecurity scenarios with the aim of enhancing the information about behavioral changes that can be related to malicious activities or cyber-attacks, such as analyzing network traffic or malware structures.

A more extensive discussion about the experiments and the obtained results is included in the following publication:

∗ Zola F., Bruse J.L., Eguimendia M., Galar M. and Orduna Urrutia R., *Bitcoin and cybersecurity: Temporal dissection of blockchain data to unveil changes in entity behavioral patterns*, Applied Sciences, 9(23), pp.5003, DOI: 10.3390/app9235003

5.2. Network traffic analysis through node behaviour classification: a graph-based approach with temporal dissection and data-level preprocessing

In this work, we present novel methodologies for addressing graph imbalance problems in disconnected graphs. Graph representations can be used to define and visualize entities' interactions and their behaviors over time. However, these graphs can easily suffer from class imbalance, i.e, when nodes/entities of a majority class are more frequent than nodes/entities of the other minority classes. In this scenario, existing graph sampling techniques tend to alter graph topology or suffer from bias toward preserving high-degree nodes. Furthermore, these techniques have shown strong limitations when disconnected graphs are used. For this reason, in this work, we introduce and compare two novel sampling approaches specially designed for disconnected graphs. These techniques are not only able to address the imbalance problem avoiding component topology changes, but can also enhance the entity classification performance.

More specifically, we introduce two novel approaches, named *R-hybrid* and *SM-hybrid*, for addressing the imbalance problem directly over the disconnected graphs. The proposed approaches exploit the disconnected structures for avoiding component topology changes. In particular,

when graph structures are used, we need to consider that not only the node behavior need to be synthesized, but also its relations with other nodes (topology). For this reason, the introduced techniques reduce the graph complexity by randomly eliminating components composed only of elements from the majority class. Then, we replicate the most "interesting" components topologies, i.e, topologies composed by at least 60 % of nodes of the minority class. In this way, all the components that satisfy the previous conditions have the same probability to be eliminated or replicated. Finally, *R-hybrid* and *SM-hybrid* differ in the way the node features in the replicated components are generated. On the one hand, *R-hybrid* assigns an entity behavior to each node of the replicated component using a ROS strategy, i.e, not only the component topology but also the node behaviors within are replicated. On the other hand, *SM-hybrid* computes the entity behavior using both ROS and SMOTE strategies. In particular, the ROS technique is used for replicating the behaviors of the nodes belonging to the majority class, whereas SMOTE technique is used for generating new synthetic node behaviors of the minority class.

In this study, these approaches are tested for addressing the imbalance problem in temporal disconnected graphs extracted from network traffic communications (or flows). In particular, a network traffic dataset that includes information about flows of normal and different attack families is used. However, for the aim of this study, we consider all attack families as a unique class to perform a binary classification. These flows are mapped into nodes/entities and edges/relations using temporal Traffic Dispersion Graphs (TDGs) [IFM09]. A temporal TDG represents a graph extracted in a temporal batch. Due to the nature of the problem, these temporal TDGs are characterized by disconnected graphs. Furthermore, since in the network traffic dataset normal activities outnumber malicious activities or attacks, the same imbalance problem is propagated to the generated disconnected graphs, where normal nodes outnumber attack ones. Similarly to the temporal graphs considered in the previous work (Section 5.1), TDGs have a strong dependency on the temporal resolution, for this reason, it is interesting to analyze how this temporal resolution affects their creation, as well as the graph population distribution (class imbalance). Finally, the choice of the ML model to be used when dealing with graph-based structures is not trivial, since methods that exploit both behavioral features and graph relations or models based only on analyzing behavioral features can be used. Therefore, we also propose to study whether using graph relations can enhance the performance with respect to only considering behavioral features.

Accordingly, we plan to address the following main questions:

1. *Do the introduced graph sampling approaches generate a more balanced dataset when disconnected graphs are used? How does the balanced information affects the classification performance?*
2. *How does the temporal resolution affect entity behavior classification?*
3. *Which is the best learning model and which configuration should be used?*

To answer these questions we analyze the goodness of our approaches by evaluating the improvements achieved in the classification of normal and attack entity behaviors after balancing the dataset. More specifically, we evaluate the performance of three supervised ML models by comparing the usage of *R-hybrid* and *SM-hybrid* with respect to not applying any strategy for the class

imbalance problem. Regarding the used supervised models, two of them are graph convolutional networks (GCNs), which can exploit both behavioral features and graph relations, whereas the other is a neural network (NN), which only considers behavioral features. In particular, the two GCNs are tested with high-order and low-order approximations for the convolution filter. We do not include common graph sampling techniques in the comparison since, as previously discussed, they are not adequate for working directly with disconnected graphs. For this reason, in order to evaluate the benefits and the limitation of the proposed approaches, we compare the obtained results with the ones achieved by 5 anomaly detection (AD) models. These models can also be appropriate to deal with highly imbalanced datasets. More specifically, three models are trained using a *semi-supervised AD* setup, whereas the other two follow an *unsupervised AD* training setup [Agg17, GU16].

To answer the second question, three different temporal resolutions (300s, 600s, and 900s) are considered and analyzed. In particular, the effect of these resolutions in the graph extraction, as well as in the classification performance, is compared. Finally, the third question is addressed by gathering and comparing the results obtained by the best configuration of the three supervised ML models. Moreover, although the whole analysis is performed as a binary classification, we also develop a study about the attack families detected by each model to extract more insights about their quality.

Due to the high complexity and high computational costs caused by all possible combinations of the configuration parameters, we fix some parameters in each question. In particular, for addressing the first question, we use a fixed value for the temporal resolution (600s) to validate the proposed approaches. Then, for the second question, the best graph sampling strategy detected is used for analyzing the influence of the other two temporal resolutions. Finally, for the third question, the best configurations, one for each supervised ML model, are compared in terms of performance in both binary and attack families detection.

Results show that both *R-hybrid* and *SM-hybrid* techniques correctly address the class imbalance problem in disconnected graphs generating a more homogenized population among normal and attack entities. Moreover, using this balanced graph dataset, all supervised ML models show improvements in their performance. In fact, models trained as semi-supervised AD together with the GCN classifiers are strongly affected by the imbalanced information, achieving very low performance if resampling is not applied. Otherwise, NN and models trained as unsupervised AD using the same imbalanced dataset, achieve promising results even if they are worse than the ones obtained by supervised ML model with balanced information. In this scenario, the best overall performance is obtained by applying *SM-hybrid* for addressing graph imbalance and implementing a GCN with a high-order approximation.

Regarding the impact of the temporal resolutions, results show that all supervised models generate their best classification results using a specific value (600s) for the given dataset. In particular, a change in the temporal resolution strongly affects the NN performance, regardless of the graph imbalance problem. In fact, performances achieved by using the balanced dataset with resolutions of 300s and 900s are lower than the ones obtained using the imbalanced dataset with resolution of 600s. Furthermore, results highlight that a low temporal resolution has a huge impact on the training time of the GCN model with a high-order approximation.

Overall, supervised models achieve their best results using the same configuration (temporal resolution of 600s and using *SM-hybrid* technique). Results highlight that graph relations can be used for improving the entity classification performance when GCN with a high-order approximation is implemented. Otherwise, using a GCN with a low-order approximation, the classification performance is negatively affected. Further analysis shows that GCN with a high-order approximation detects seven out of eight attack families with very high accuracy. Nevertheless, two concrete attack families are detected more precisely by using the NN, whereas all the models fail in detecting a (single) specific family.

According to the results of this work, the following conclusions can be highlighted:

- Both introduced techniques (*R-hybrid* and *SM-hybrid*) correctly address the graph imbalance problem when disconnected graphs are used. Both techniques are able to create a balanced population between normal and attack entities. Furthermore, considering entire components' structure at once helps them to not modify component relations (topology) while avoiding changes in the untouched components.
- *R-hybrid* and *SM-hybrid* allow one to improve entity classification performance, especially for the GCNs. Among both methods, models trained on datasets preprocessed with *SM-Hybrid* achieved the highest classification performance, highlighting that new synthetic behaviors help the ML models to enhance their learning process.
- The classification performance can be improved considering graph relations if a high-order approximation is used. However, this comes at the cost of higher complexity and computational efforts, increasing training times.
- Finally, in this study, establishing the temporal resolution has shown to be a key factor to properly differentiating normal and attack classes. Using the lowest resolution may not allow the behaviors to be properly represented, whereas using the highest resolution may produce behaviors to be overlooked by more complex macro-dynamics

A deeper discussion about results and conclusion, as well as a comparison between our classification approaches and previously published models, is reported in the following publication:

∗ Zola F., Segurola Gil L., Bruse J.L., Galar M. and Orduna Urrutia R., *Network traffic analysis through node behaviour classification: a graph-based approach with temporal dissection and data-level preprocessing*, Computers & Security, 115 (2022): 102632, DOI: 10.1016/j.cose.2022.102632

5.3. Attacking Bitcoin anonymity: Generative Adversarial Networks for improving Bitcoin entity classification

In this work, we propose a study about how Generative Adversarial Networks (GANs) can be used as an alternative to the traditional data-level preprocessing techniques for addressing

class imbalance problems when cybersecurity tabular data are involved. In fact, common data-level preprocessing methods can generate downside effects such as underfitting and overfitting, or generate samples with low variability. In this scenario, our hypothesis is that the usage of GANs can unveil more information in the dataset, generating more variable samples that can help to better explore the feature space, increasing classifier abilities. However, as introduced in Section 2.4, many GANs architectures are available, since a unique solution that works in every scenario does not exist. For this reason, we implement three of the main GAN architectures and use them for generating synthetic samples and addressing the imbalance problem in cybersecurity tabular data. Finally, we evaluate the quality of the generated synthetic samples evaluating their impact on the classification performance.

More specifically, we use GANs to model the real data distribution and then perform an over-sampling using the synthetic data generated by GANs. Our idea is to train a GAN for each under-represented (minority) class. Then, each generator of the GAN is used to generate synthetic data so that each class reaches the population of the majority one. This operation creates an enriched dataset formed of synthetic and real data that is used to train the final classifier. In this way, it is possible to evaluate the quality of the synthetic samples based on the entity classification performance. In particular, we propose to study three of the main GAN architectures, known as *Vanilla GAN, Wasserstein GAN (WGAN)* and *unrolled GAN*. The first architecture is selected for its simplicity and flexibility in the implementation, representing a reference solution. The second architecture (*WGAN*) is chosen for its ability to improve the model stability and make the training process easier [ACB17]. Finally, the third architecture (*unrolled GAN*) is selected as it allows one to improve the GAN dynamics by bringing the discriminator closer to an optimal response [MPPSD17]. These GAN architectures create a good benchmark set for evaluating the benefits and limitations of this technology when cybersecurity tabular data are used. However, the quality of the samples generated by a GAN is strongly related to the number of epochs used to train the GAN itself. In this sense, setting improper parameters for training the GAN can produce downside effects such as *non-convergence, vanishing gradient,* or *mode collapse* (Section 2.4), hindering the quality of the generated samples. At the same time, we need to be sure that the generated samples and the effect they generate in the classification, do not represent outlier solutions. Finally, since we aim to use GANs to create synthetic samples for balancing the initial dataset, this approach must be compared with traditional data-level techniques to evaluate its usefulness.

In this study, the introduced GAN strategy for addressing class imbalance is validated using a Bitcoin blockchain dataset. A similar dataset was also used in Section 5.1 for analyzing temporal behavior consistency. However, in this case, our main goal is to use GANs for generalizing Bitcoin entity behaviors and generating new synthetic samples, which address the Bitcoin imbalance problem. In this way, the GAN-balanced dataset is expected to enhance the model performance in the entity classification task, where six different entity behaviors are considered: *Exchange, Gambling, Marketplace, Mining Pool, Mixer* and *Service*. These behaviors are characterized by the nodes' interactions within the address-transaction graph (Section 2.2.2). Nevertheless, the gathered dataset suffers from class imbalance, since it is easier to gather information about entities related to licit activities (Exchange, Gambling) than about entities involved in illicit ones (Mixer,

Service).

In this scenario, we plan the following main questions:

1. *Do GAN strategies address the Bitcoin imbalance problem and enhance the entity behavior classification? Which is the best GAN architecture and how does the number of epochs used in the training* affect its learning process?

2. *Are the results obtained by the best GAN configuration repeatable (randomness analysis)?*

3. *Could the GAN strategy represent an alternative to common data-level preprocessing techniques?*

In order to answer the first question, we compare the classification performance obtained by classifiers in which the imbalance problem is addressed using the three GAN architectures with a classifier that uses directly the imbalanced information (baseline model). Moreover, for evaluating the effect of the number of epochs in the GAN learning process, for each architecture, we consider six different training checkpoints. A training checkpoint represents a fixed number of epochs in which we stop the GAN training. At each stop, the GAN generators are used to create a balanced version of the initial dataset, which in turn is used to train a new entity behavior classifier. This approach let us analyze the relation between the GAN architecture, the number of epochs, and the quality of the generated samples.

After validating the approach, the second question is addressed by analyzing the repeatability of the results of the best solution. In this sense, the best GAN configuration previously detected is used to train five GANs for each class sharing the hyper-parameters, which are then used to implement five entity behavior classifiers with artificially balanced datasets. This approach helps us to analyze and evaluate the repeatability of the results obtained by using a specific GAN configuration.

Finally, the third question is addressed by comparing the results obtained by the introduced GAN-based approach with results obtained applying five traditional data-level preprocessing techniques (RUS, ROS, TL, SMOTE, and ADASYN).

Overall, improvements in the classification performance are obtained by addressing the class imbalance problem using the introduced GAN approach. More specifically, overall enhancements are obtained regardless of the used GAN architecture and the number of training epochs, although the *Vanilla* and *unrolled GANs* perform better using fewer training epochs, whereas *WGAN* require more epochs for obtaining better synthetic samples. Analyzing the model performance per class, all the classifiers that use the datasets balanced by the *Vanilla* and *unrolled GANs* show improvements in detecting all the entities, regardless of the number of epochs used. On the other hand, classifiers trained with datasets balanced by the *WGAN* show a deterioration in the Mining Pool detection. The best overall performance is achieved by the classifier trained with the dataset balanced by the *Vanilla GAN* trained for a few epochs (1,000).

The best solution also shows a high repeatability in its results. In fact, repeating the GAN training five more times with as many classifiers, very low performances deviations are obtained.

The study shows also that our GAN-based approach generates improvements in the overall classification performance better than other techniques such as RUS, TL, SMOTE, and ADASYN. However, they are slightly lower than the ones generated by the ROS strategy. This is mainly due to the low quality of the GAN samples generated for two specific classes (Mining Pool and Service). In fact, for these classes, the presence of dense regions in the training data distribution lead GAN to learn and generate only partial information, hence limiting its performance. Finally, in terms of computational costs, excluding the two random resampling techniques (RUS and ROS), our GAN approach is the most efficient strategy. To complement this analysis we performed an additional comparison with previous works on Bitcoin entity classification. In particular, our classifier learned using the GAN balanced dataset is the best in classifying two classes (Exchange and Marketplace) and is the second-best solution for other two (Gambling and Service). However, it shows problems in detecting the last two classes (Mining Pool and Mixer).

Based on the results obtained in this work, the following conclusions are highlighted:

- Class imbalance problem should be tackled for improving the quality of the classification performance. GAN solutions are a promising alternative to traditional resampling techniques also when cybersecurity tabular data are used. In particular, they show the ability to learn and generalize behavioral entities which in turn helps them in generating high-quality samples, improving the final entity behavior classification. Our study shows that overall improvements in the classification are obtained regardless of the GAN architecture used. Among the tested architectures, the best results are achieved using a *Vanilla GAN*.

- The number of training epochs affects the quality of the samples in different ways according to the GAN architecture. In fact, we have found that *Vanilla GAN* and *unrolled GAN* perform better when trained with fewer epochs, whereas *WGAN* requires a larger number of epochs. Hence, the GAN training duration needs to be carefully considered in each architecture and use case.

- The learning process of GANs depends on the input data distribution. Dense areas in the feature space can lead GANs to fail their task, since they can focus on learning just certain behaviors, generating only partial information (recalling the *mode collapse* effect). In our case study, this phenomenon has been identified in two specific classes (Mining Pool and Service)

- Generative models can be effectively used to recreate activities carried out by real Bitcoin users to discover new patterns and prevent unknown cyber-threats.

The complete information on this work has been published in the following article:

∗ Zola F., Segurola Gil L., Bruse J.L., Galar M. and Orduna Urrutia R., *Attacking Bitcoin anonymity: Generative Adversarial Networks for improving Bitcoin entity classification*, Appl Intell (2022), DOI: 10.1007/s10489-022-03378-7

5.4. Cyber Threat Intelligence for Malware Classification in the Presence of Concept Drift

In this work, we present a novel methodology for enhancing CTI based on temporal analysis, concept drift, and an in-depth misclassification investigation. The main idea is to shed light on the underlying issues related to cybersecurity ML-based classification and investigate potential causes of model failures. In this way, it is possible to extract insightful information relevant to cybersecurity practitioners and analysts. In fact, in recent years, in cybersecurity applications, ML algorithms have been mainly used for modelling the input data [MTICGN19] and improving the performance of classification tasks, especially in malware analysis [UAB19,SH18], becoming a race toward the best classification performance while neglecting the interpretation of the results. Being so focused on performance improvement, it is common to overlook issues that can lead to overly optimistic conclusions that do not reflect real-world cybersecurity scenarios [AQP+22]. Additionally, in several cases, conclusions are drawn from laboratory settings, i.e., "static" environments, which do not represent the real world in which behaviours and patterns are continuously evolving over time [AQP+22]. These dynamics in the data may cause trained ML models to become outdated after only a few months without being noticed [PPJ+19]. This situation can result in a loss of control over the model behavior, especially knowing that usually the ML models are trained by experts that are neither the actual cybersecurity operators nor analysts [SBE19].

In this dissertation, we introduce a methodology for extracting insightful information related to data distribution and concept drifts that can affect ML models in cybersecurity and enhance CTI. More specifically, we focus the analysis on malware classification using Portable Executables (PEs) of 6 different families (Adware, Dropper, Spyware, File Infector, Worm, and Downloader) gathered from January 2017 to April 2019 (28 months) and included in the SOREL-20M dataset [HR20]. To evaluate the model performance, its evolution over time and concept drift, we consider the F1-score and the Area Under Time (AUT) [PPJ+19], which is the area formed by the F1-scores obtained over time. Our methodology is based on the following three steps:

1. **Classification.** In this step, we first convert the PE malwares to CFGs and then, we use 10 common graph properties (feature embeddings) to describe each CFG. These embeddings are split considering the temporal factor, i.e., knowing that the whole dataset contains information of 28 months. The first 18 are used for the training dataset, the following 5 for the validation dataset and the last 5 months for creating the test dataset. These datasets are finally used for training, validating, and testing ML models based on Random Forest. More specifically, a grid-search over the validation set is performed to tune the values regarding the number of trees and the tree depth. Afterwards, we used the test set to evaluate the generalization capabilities of the model and check whether the temporal difference causes a significant drop in performance compared to validation metrics.

2. **Temporal analysis.** We propose a temporal analysis based on two main concepts: temporal dissection and temporal aggregation. In temporal dissection, as introduced in Section 5.1, we evaluate how the behaviors change over time and how they affect the classification performance using a rolling window approach. In this sense, we divide the initial dataset into chunks of 1 month. Then, the rolling window is used to keep selecting two consecutive

months for training a classifier, whereas the following months are used for testing it. The process is repeated by moving the rolling window over the available dataset. On the other hand, the temporal aggregation changes the size of the training dataset to incorporate more information. In particular, we apply the temporal aggregation for training 5 different models, using training datasets composed of 2, 4, 6, 12, and 18 months. This approach simulates training with as much data as one has at hand in an equivalent real-world scenario, and it helps to detect consistent performance drops, i.e., drops found even when larger datasets are used during the model training. These points can be related to concept drifts and must be further analyzed.

3. **Misclassification and feature trends.** Once drops are detected, we analyze their possible causes by focusing on the misclassification results and temporal trends of the input features. In this case, we first extract and analyze the confusion matrices of the models on the dates on which they show the performance drop. We restrict our feature trend analysis to the most important features, which are found using the Mean Decrease Impurity (MDI) [Sco20]. The trend of these features is extracted and observed in the performance drop points, looking for abrupt changes and rare behaviors that may help to explain what the model learned and why it misclassified certain classes.

These three steps are tested in the malware classification problem. In the first step, all the tested models reach promising values in terms of F1-score in the validation set ($\geq 82\,\%$). The results highlight that the depth of the tree is the parameter that allows improving the performance of the classifier, whereas the number of trees predominantly has an impact on the training time. However, when evaluating the generalization of the model with the test set, we find a loss of performance with respect to the validation set. This can be due to how the malware families' distribution changes over time. In fact, although a balanced dataset is considered, several malware families (Dropper and Spyware) have more samples in the validation dataset than in the train dataset, whereas others (Worm) have more samples in the test dataset than in the train and validation sets, due to the temporal data partitioning scheme used. Therefore, a temporal analysis should be performed to investigate how model performance evolves over time and identify performance drop points.

In the second step, the temporal dissection shows that models trained with the early months in the dataset generate very poor results in terms of F1-score in almost every test month. However, when more recent months are used, the models show immediate high values in terms of F1-sore for the next 5 months, but they worsen their classification performance afterwards, hinting at the presence of concept drift. On the other hand, the temporal aggregation shows that the model trained with just the first 2 months achieves very low values in all the considered metrics, whereas the model trained with 18 months shows higher values. This trend apparently confirms that the larger the training dataset is, the better the overall classifier performance is. Nevertheless, the other models, trained with 4, 6, and 12 months, show highly varying trends depending on the considered dates. In this sense, comparing the performance between the models trained for only 2 months (dissection models) and their respective models trained with all the data available until those months (aggregation models), it is to be noted that in several cases, dissection models can

be competitive, i.e., including more data into the training process do not generate substantial improvements. Furthermore, models trained with fewer data show a *gradual* concept drift, since they present a first strong decline and a subsequent loss of performance. On the other hand, models trained with more data do not show a clear concept drift pattern, even if, on specific dates, they show strong performance drops. These drop points can be considered as an *abrupt* concept drift or *outliers*. More specifically, three abrupt drops were further analyzed to detect their possible causes, since they were present even when models were trained with larger datasets.

Therefore, in the third step, we analyze the three performance drop points. First, we study the confusion matrices of the models. In this way, we focus the considerations and conclusions just over the confused classes. In the first drop point, Dropper elements are mainly confused with Spyware and File Infector, whereas in the second and third drop points, they are confused with Adware and Downloader samples, and Spyware elements are confused with Worm and File Infector. Then, to analyze the behavior of the input features over time for the confused classes, we identify the most important features using MDI. The feature importance analysis shows that 9 out of 10 of the used structural graph properties extracted from the CFG are used by all the models for the final classification, although each one with a different importance score. This analysis is important to understand which property represents a sensitive target and hence, the most prone to be used in an attack. In fact, a slight change in its value has a greater influence on misclassification. The top-4 features with the highest importance scores in all the implemented models are *transitivity, number of strongly connected components, average degree of nodes* and *number of nodes*. For these features, we analyze their temporal trend and observed that usually there is at least one feature that changed substantially in the drop points. More specifically, the values of these features for the confused classes seem to line up, only in the model failure dates. Probably this alignment of such features leads to misclassifications.

Based on the results obtained in this work, the following conclusions are highlighted:

- Concept drift and several distinct points of model failure should be analyzed in order to identify the limitations of the models, even for models trained on a relatively large amount of data.
- The temporal dissection approach highlighted that several models trained with only 2 months of recent data performed accurately in months closer to the months used for training them, whereas their performance dropped drastically when evaluating data from about 5 months later, showing concept drifts and distinct points of model failure. This highlight that, especially in real-world applications, models have a "lifetime" and they should be re-trained throughout their life cycle.
- The temporal aggregation approach helps to discover model limitations and model failure points that also affect models trained with a relatively large amount of data. The obtained results showed that considering always the largest amount of data in the training dataset for improving the classification performance is not enough. In fact, these models also show performance drops, and in several cases, their performance in the classification of the closest temporal samples does not generate substantial improvements with respect to the ones obtained using models trained with fewer data.

- Analyzing the trends of the most important features over time for the confused classes, it is possible to extract novel information for relating input trends to critical points of model failure. In our case study, model failures are linked to substantial changes in at least one input feature. This information could be used by cyber analysts and experts to train adaptative ML models able to evolve together with the input data, limiting performance decays.

The complete work corresponding to this part can be found in the following article:

∗ Zola F., Bruse J.L., Galar M. and Cavallaro L., *Cyber Threat Intelligence for Malware Classification in the Presence of Concept Drift*, Submitted.

6. Conclusions

In this section, we report the outcomes of this thesis focused on behavioral analysis in Cybersecurity using Machine Learning. The study is based on graph representation, class imbalance, and temporal dissection. The conclusions are drawn from the different works previously described.

In cybersecurity applications, time-series information could be not enough to describe behaviors of complex systems in which entities and relations evolve over time. Therefore, in this thesis, we focused on using a graph-based representation that can be used for highlighting network dynamics favoring the classification and visualization of such information. However, in this dynamic scenario, it is relevant to analyze how the data temporality affects the problem description, i.e., how it affects the graph representation, behavior definitions, and consequently their classification. For this reason, specific analysis needs to be performed depending on the domain application. In this thesis, we considered two different operations: *temporal dissection* and *temporal aggregation*. The first strategy is used for highlighting micro-dynamics in the graphs and evaluating how entity behaviors change over time, whereas the second is used for detecting drifts in the classification performance.

Cybersecurity behavioral analysis can be performed directly using the graph representation or using tabular data (features or behaviors) extracted from the graph. However, in both cases, class imbalance represents a very challenging problem that affects the quality of the predictions. In this thesis, we firstly analyzed and addressed the class imbalance directly in the graphs. In particular, we implemented two novel strategies able to exploit disconnected graphs for addressing the imbalance. Both strategies are drawn to avoid component topology modification. On the other hand, we presented a solution based on GANs for working with the tabular data extracted from a graph. In this case, the GANs are used for learning complex behaviors and for generating new synthetic ones, which are finally used for addressing the imbalance in the tabular dataset, as an oversampling technique.

In this thesis, we did not limit the analysis only to the performance evaluation, but we aimed to highlight the problems that may arise when ML models are used for real-world cybersecurity tasks. In this sense, we used CTI concepts and mechanisms for discovering new insights that

cyber analysts and operators can use during their tasks for increasing their domain knowledge. In particular, we considered the misclassification, data trends, and feature importance information for investigating potential causes of model failure and for shedding light on the model decisions.

In summary, the main conclusions extracted from this thesis are the following:

- Graph representations allow one to obtain a different and value-adding perspective in cybersecurity behavioral analyses. This approach showed to improve the state-of-the-art results, especially in classification tasks where graphs were used for representing complex systems such as blockchain networks, communications networks, and malware binaries.
- Data temporality affects differently the input distribution and the classification performance. More specifically, *temporal dissection* allows for checking the consistency of the entity behaviors over time, and it highlights micro-dynamics in the graph. On the other hand, *temporal aggregation* allows one to detect eventual drifts in the classification performance.
- Two novel solutions for addressing the graph imbalance problem using disconnected graphs are implemented. Both techniques allow one to create a more balanced population in the graph structure avoiding component topologies modification. Moreover, the usage of these balanced graphs generated improvement in the behavioral classification performance.
- GANs allow one to learn complex entity behaviors and create synthetic ones that can be used for addressing the imbalance problem, as an oversampling technique. This thesis showed that the results of GANs depend on the chosen architecture and the number of epochs used for training it. Furthermore, although the GAN approach enhances the classification performance, these improvements are still lower than the results obtained by using the ROS strategy.
- CTI can be used to relate misclassification and feature trends and highlight patterns that may generate concept drift over time. In this sense, for helping cyber analysts and experts in their task and for avoiding unexpected drifts, it is better to train models on fewer but recent data, apply them for a few months only, and then re-train the model using the newest data again.

7. Future works

We conclude the first part of this thesis by drawing guidelines and future works that can be developed from the presented outcomes.

To replicate temporal behaviors in a controlled environment for discovering new unknown behavioral patterns

As presented in this thesis, entity behaviors are continuously evolving over time creating micro-dynamics that are hard to be detected. In this sense, the high variability of the samples is usually not properly represented in the training dataset, limiting the performance of the classification models. Furthermore, in several applications such as blockchain transactions, entity behaviors are

strictly related among them due to action-reaction links. In fact, a behavior not only depends on the generated transactions of the specific entity, but also on the received transactions generated by the others. In this complex and multi-related scenario, it is interesting to be able to simulate such relations in a controlled environment. More specifically, in the case of blockchain-related behaviors, the testbed presented in [ZPSZ+19] could be used for deploying a private blockchain network and generating transactions among its participants. This approach could be used for replicating and simulating known (typical) behaviors, specific types of attacks, or illicit activities related to certain entities allowing us to dissect all the small interactions and highlight which transactions are the most relevant for defining the entity behavior. Furthermore, this synthetic information could be directly included in real-world models to validate how the simulated behaviors recall the real ones.

To develop new graph sampling techniques based on node behavior for decreasing the computational cost and improving the effectiveness of the analysis

As shown in this thesis, using graph representation in the cybersecurity domain generates promising results, regardless of the nature of the data. However, we need to deal with very large graphs that in several cases can require high computational costs. In this sense, it would be interesting to reduce the graph representation without losing relevant information by proposing new graph sampling techniques. An interesting solution could be based on aggregating similar node behaviors to reduce the overall information. This operation could have a strong impact, especially when the initial dataset is characterized by a huge number of similar nodes or by noisy data. For example, density-based clustering could be considered. This cluster approach uses unsupervised learning methods to identify contiguous regions of high point density (clusters) and regions of low point density (noise/outliers) [KKSZ11]. Once similar behaviors were detected, they could be merged into a unique behavior that represents all of them. Hence, the information to be analyzed would be reduced, reducing the computational costs. This approach could be also integrated in both *R-hybrid* and *SM-hybrid* strategies introduced in Section 5.2, replacing the RUS operation. However, the impact that this modification would generate in addressing the graph imbalance problem and in the final analysis, should be explored.

To study the usefulness of inductive embeddings for codifying graph/nodes and improving the final performance of the classifiers

Regarding the classification task, in this thesis, we have analyzed two approaches. On the one hand, we directly used real-world features and graph topology for training graph ML techniques (Section 5.2). On the other hand, we extracted and used graph properties (or node behaviors) as input for traditional ML models (Section 5.1 and Section 5.4). However, in future works, it would be interesting to apply techniques able to convert high-dimensional sparse graphs into low-dimensional and continuous vector spaces (embeddings). This approach would help to reduce high computational costs and excessive memory usage, as well as obtain heterogeneous characteristics that can preserve graph properties improving the final performance. More specifically, it would be interesting to analyze a specific class of embedding called *inductive*. This category includes techniques that are able to predict the embedding of unseen nodes, unlike the *transductive* which can only work with nodes seen during the training phase [HYL17]. This strategy could be used for enhancing different graph analytics tasks such as node classification, graph similarity, link prediction, visualization, etc. In particular, inductive techniques in cybersecurity applications

could be studied in terms of properties, i.e., which embedding provides a better representation of nodes/graph; scalability, i.e., how they can preserve the properties when huge graphs are analyzed; and finally in terms of optimal dimensionality, i.e., finding optimal dimensions of the representation of the graph (according to the application domain).

To favor the explainability of the results by relating CTI information with real-world data structures

The CTI methodology introduced in this thesis (Section 5.4) allows one to extract information about how the performance of a malware classifier changes over time. The approach highlighted that malware classifiers need to be frequently retrained (after a couple of months) to avoid drifts, since malware binaries are continuously evolving changing their CFG representation. In this sense, this thesis showed how the performance decay can be related to several CFG properties that were used as inputs of the classifier. However, in future works, it would be interesting to relate the performance decay directly to the malware calls and routines, i.e., the nodes of the CFG. In fact, in this way, it would be possible to understand which part of the CFG is the most relevant during the classification, highlighting which part of the CFG is typical of each malware family. This approach would help cyber analysts and operators to better explain and understand the decision taken by the ML model. On the other hand, once specific structures for each malware family were defined, other binaries such as firmware, drivers, and more complex codes, could be analyzed for discovering structural similarities between them and malware, which can be a hint of a vulnerability. Finally, it must be taken into account that the explainability of the results, i.e., the knowledge about relations between model decisions and real-world data structures, can also be used by hackers and malicious actors to change the internal structure of their malware for creating more “intelligent” threats, which can mislead the detector. For this reason, a CTI methodology for sharing and protecting this sensitive information should be proposed and validated.

Bibliography

[AA18] Alsehibani S. y Almuhammadi S. (2018) Anomaly detection: Firewalls capabilities and limitations. In *2018 International Conference on Computing Sciences and Engineering (ICCSE)*, pages 1–5. IEEE.

[AAD+21] Abbas K., Abbasi A., Dong S., Niu L., Yu L., Chen B., Cai S.-M., y Hasan Q. (2021) Application of network link prediction in drug discovery. *BMC bioinformatics* 22(1): 1–21.

[AB17] Arjovsky M. y Bottou L. (2017) Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862* .

[ACB17] Arjovsky M., Chintala S., y Bottou L. (06–11 Aug 2017) Wasserstein generative adversarial networks. In Precup D. y Teh Y. W. (Eds.) *Proceedings of the 34th International Conference on Machine Learning*, volumen 70, pages 214–223. PMLR.

[Agg17] Aggarwal C. C. (2017) An introduction to outlier analysis. In *Outlier analysis*, pages 1–34. Springer.

[AHA20] Athanasiadis C., Hortal E., y Asteriadis S. (2020) Audio–visual domain adaptation using conditional semi-supervised generative adversarial networks. *Neurocomputing* 397: 331–344.

[ALPA17] Ahmad S., Lavin A., Purdy S., y Agha Z. (2017) Unsupervised real-time anomaly detection for streaming data. *Neurocomputing* 262: 134–147.

[AMKM17] Abid A., Masmoudi A., Kachouri A., y Mahfoudhi A. (2017) Outlier detection in wireless sensor networks based on optics method for events and errors identification. *Wireless Personal Communications* 97(1): 1503–1515.

[ANG+18] Agrafiotis I., Nurse J. R. C., Goldsmith M., Creese S., y Upton D. (2018) A taxonomy of cyber-harms: Defining the impacts of cyber-attacks and understanding how they propagate. *J. Cybersecur.* 4: tyy006.

[AOC07] Ahmed T., Oreshkin B., y Coates M. (2007) Machine learning approaches to network anomaly detection. In *Proceedings of the 2nd USENIX workshop on Tackling computer systems problems with machine learning techniques*, pages 1–6. USENIX Association.

[AQP+22] Arp D., Quiring E., Pendlebury F., Warnecke A., Pierazzi F., Wressnegger C., Cavallaro L., y Rieck K. (2022) Dos and don'ts of machine learning in computer security. In *Proc. of the USENIX Security Symposium*.

[ATK15] Akoglu L., Tong H., y Koutra D. (2015) Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery* 29(3): 626–688.

[Bar12] Barnum S. (2012) Standardizing cyber threat intelligence information with the structured threat information expression (stix). *Mitre Corporation* 11: 1–22.

[BDS18] Brock A., Donahue J., y Simonyan K. (2018) Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations*.

[Ber84] Berge C. (1984) *Hypergraphs: combinatorics of finite sets*, volumen 45. Elsevier.

[BGCML21] Blázquez-García A., Conde A., Mori U., y Lozano J. A. (2021) A review on outlier/anomaly detection in time series data. *ACM Computing Surveys (CSUR)* 54(3): 1–33.

[Bol13] Bollobás B. (2013) *Modern graph theory*, volumen 184. Springer Science & Business Media.

[C+07] Council N. R. *et al.* (2007) *Toward a safer and more secure cyberspace*. National Academies Press.

[CBK09] Chandola V., Banerjee A., y Kumar V. (2009) Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41(3): 1–58.

[CC19] Chalapathy R. y Chawla S. (2019) Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407* .

[CDD18] Conti M., Dargahi T., y Dehghantanha A. (2018) Cyber threat intelligence: challenges and opportunities. In *Cyber Threat Intelligence*, pages 1–6. Springer.

[CFQS12] Casteigts A., Flocchini P., Quattrociocchi W., y Santoro N. (2012) Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems* 27(5): 387–408.

[CHZ13] Cupertino T. H., Huertas J., y Zhao L. (2013) Data clustering using controlled consensus in complex networks. *Neurocomputing* 118: 132–140.

[CJR+19] Coley C. W., Jin W., Rogers L., Jamison T. F., Jaakkola T. S., Green W. H., Barzilay R., y Jensen K. F. (2019) A graph-convolutional neural network model for the prediction of chemical reactivity. *Chemical science* 10(2): 370–377.

[CKA+17] Chowdhury S., Khanzadeh M., Akula R., Zhang F., Zhang S., Medal H., Marufuzzaman M., y Bian L. (2017) Botnet detection using graph-based feature clustering. *Journal of Big Data* 4(1): 1–23.

[CKBR06] Carl G., Kesidis G., Brooks R. R., y Rai S. (2006) Denial-of-service attack-detection techniques. *IEEE Internet computing* 10(1): 82–89.

[CMF19] Cook A. A., Mısırlı G., y Fan Z. (2019) Anomaly detection for iot time-series data: A survey. *IEEE Internet of Things Journal* 7(7): 6481–6494.

[CR15] Chismon D. y Ruks M. (2015) Threat intelligence: Collecting, analysing, evaluating. *MWR InfoSecurity Ltd. https://www.foo.be/docs/informations-sharing/Threat-Intelligence-Whitepaper.pdf* .

[CWH18] Crawford F. W., Wu J., y Heimer R. (2018) Hidden population size estimation from respondent-driven sampling: a network approach. *Journal of the American Statistical Association* 113(522): 755–766.

[CZ18] Carneiro M. G. y Zhao L. (2018) Analysis of graph construction methods in supervised data classification. In *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 390–395. IEEE.

[CZC18] Cai H., Zheng V. W., y Chang K. C.-C. (2018) A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering* 30(9): 1616–1637.

[Dal14] Dalziel H. (2014) *How to define and build an effective cyber threat intelligence capability*. Syngress.

[DB18] Douzas G. y Bacao F. (2018) Effective data generation for imbalanced learning using conditional generative adversarial networks. *Expert Systems with applications* 91: 464–471.

[DBV16] Defferrard M., Bresson X., y Vandergheynst P. (2016) Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852.

[DCD+18] Dehghantanha A., Conti M., Dargahi T., *et al.* (2018) *Cyber threat intelligence*. Springer.

[DGAH+21] Darem A. A., Ghaleb F. A., Al-Hashmi A. A., Abawajy J. H., Alanazi S. M., y Al-Rezami A. Y. (2021) An adaptive behavioral-based incremental batch learning malware variants detection model using concept drift detection and sequential deep learning. *IEEE Access* 9: 97180–97196.

[DPBC+15] Dal Pozzolo A., Boracchi G., Caelen O., Alippi C., y Bontempi G. (2015) Credit card fraud detection and concept-drift adaptation with delayed supervised information. In *2015 international joint conference on Neural networks (IJCNN)*, pages 1–8. IEEE.

[DSSVW11] Djidjev H., Sandine G., Storlie C., y Vander Wiel S. (2011) Graph based statistical analysis of network traffic. In *Proceedings of the Ninth Workshop on Mining and Learning with Graphs*.

[FE09] Fenz S. y Ekelhart A. (2009) Formalizing information security knowledge. In *Proceedings of the 4th international Symposium on information, Computer, and Communications Security*, pages 183–194.

[FGG+18a] Fernández A., García S., Galar M., Prati R. C., Krawczyk B., y Herrera F. (2018) Algorithm-level approaches. In *Learning from Imbalanced Data Sets*, pages 123–146. Springer.

[FGG+18b] Fernández A., García S., Galar M., Prati R. C., Krawczyk B., y Herrera F. (2018) *Learning from imbalanced data sets*, volumen 11. Springer.

[FGHC18] Fernandez A., Garcia S., Herrera F., y Chawla N. V. (2018) Smote for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. *Journal of artificial intelligence research* 61: 863–905.

[FHOM09] Ferri C., Hernández-Orallo J., y Modroiu R. (2009) An experimental comparison of performance measures for classification. *Pattern recognition letters* 30(1): 27–38.

[FHR18] Feng W., Huang W., y Ren J. (2018) Class imbalance ensemble learning based on the margin theory. *Applied Sciences* 8(5): 815.

[FKP15] Fleder M., Kester M. S., y Pillai S. (2015) Bitcoin transaction graph analysis. *arXiv preprint arXiv:1502.01657* .

[FMS13] Flocchini P., Mans B., y Santoro N. (2013) On the exploration of time-varying networks. *Theoretical Computer Science* 469: 53–68.

[FRL+17] Fedus W., Rosca M., Lakshminarayanan B., Dai A. M., Mohamed S., y Goodfellow I. (2017) Many paths to equilibrium: Gans do not need to decrease a divergence at every step. *arXiv preprint arXiv:1710.08446* .

[GBR+17] Gomes H. M., Bifet A., Read J., Barddal J. P., Enembreck F., Pfharinger B., Holmes G., y Abdessalem T. (2017) Adaptive random forests for evolving data stream classification. *Machine Learning* 106(9): 1469–1495.

[GC17] Gu Y. y Cheng L. (2017) Classification of class overlapping datasets by kernel-mts method. *International Journal of Innovative Computing, Information and Control* 13(5): 1759–1767.

[GCC20] Goyal P., Chhetri S. R., y Canedo A. (2020) dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems* 187: 104816.

[GCZ21] Gao H., Cheng S., y Zhang W. (2021) Gdroid: Android malware detection and classification with graph convolutional network. *Computers & Security* 106: 102264.

[GFB+11] Galar M., Fernandez A., Barrenechea E., Bustince H., y Herrera F. (2011) 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.

[Goo16] Goodfellow I. (2016) Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160* .

[GPAM+14] Goodfellow I., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., y Bengio Y. (2014) Generative adversarial nets. *Advances in neural information processing systems* 27.

[GS17] Gosain A. y Sardana S. (2017) Handling class imbalance problem using oversampling techniques: A review. In *2017 international conference on advances in computing, communications and informatics (ICACCI)*, pages 79–85. IEEE.

[GU16] Goldstein M. y Uchida S. (2016) A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS one* 11(4): e0152173.

[GYD+08] Guo X., Yin Y., Dong C., Yang G., y Zhou G. (2008) On the class imbalance problem. In *Fourth international conference on natural computation*, volumen 4, pages 192–201. IEEE.

[Ham20] Hamilton W. L. (2020) Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning* 14(3): 1–159.

[HL13] Hu P. y Lau W. C. (2013) A survey and taxonomy of graph sampling. *arXiv preprint arXiv:1308.5865* .

[HML+14] Han W., Miao Y., Li K., Wu M., Yang F., Zhou L., Prabhakaran V., Chen W., y Chen E. (2014) Chronos: a graph engine for temporal graph analysis. In *Proceedings of the Ninth European Conference on Computer Systems*, pages 1–14.

[HMZ+17] Hu D., Ma Z., Zhang X., Li P., Ye D., y Ling B. (2017) The concept drift problem in android malware detection and its solution. *Security and Communication Networks* 2017.

[HOFFR12] Hernández-Orallo J., Flach P., y Ferri Ramírez C. (2012) A unified view of performance metrics: Translating threshold choice into expected classification loss. *Journal of Machine Learning Research* 13: 2813–2869.

[HR20] Harang R. y Rudd E. M. (2020) Sorel-20m: A large scale benchmark dataset for malicious pe detection.

[HS12] Holme P. y Saramäki J. (2012) Temporal networks. *Physics reports* 519(3): 97–125.

[HWH20] Han J., Woo J., y Hong J. W.-K. (2020) Oversampling techniques for detecting bitcoin illegal transactions. In *21st Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 330–333. IEEE.

[HYL17] Hamilton W., Ying Z., y Leskovec J. (2017) Inductive representation learning on large graphs. *Advances in neural information processing systems* 30.

[HZS+20] He J., Zheng J., Shen Y., Guo Y., y Zhou H. (2020) Facial image synthesis and super-resolution with stacked generative adversarial network. *Neurocomputing* .

[IBN+15] Iannacone M., Bohn S., Nakamura G., Gerth J., Huffer K., Bridges R., Ferragut E., y Goodall J. (2015) Developing an ontology for cyber security knowledge graphs. In *Proceedings of the 10th Annual Cyber and Information Security Research Conference*, pages 1–4.

[IFM09] Iliofotou M., Faloutsos M., y Mitzenmacher M. (2009) Exploiting dynamicity in graph-based traffic analysis: Techniques and applications. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 241–252.

[Int21] Interpol (2021) National cybercrime strategy guidebook. *https://www.interpol.int/content/download/16455/file/National %20Cybercrime %20 Strategy %20Guidebook.pdf.* Accessed 01 May 2022.

[IPF+07] Iliofotou M., Pappu P., Faloutsos M., Mitzenmacher M., Singh S., y Varghese G. (2007) Network monitoring using traffic dispersion graphs (tdgs). In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 315–320.

[JCG+19] Jiang J., Chen J., Gu T., Choo K.-K. R., Liu C., Yu M., Huang W., y Mohapatra P. (2019) Anomaly detection with graph convolutional networks for insider threat and fraud detection. In *MILCOM 2019-2019 IEEE Military Communications Conference (MILCOM)*, pages 109–114. IEEE.

[JDW+21] Jin W., Derr T., Wang Y., Ma Y., Liu Z., y Tang J. (2021) Node similarity preserving graph convolutional networks. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 148–156.

[JGH+20] Jin B., Gao C., He X., Jin D., y Li Y. (2020) Multi-behavior recommendation with graph convolutional networks. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 659–668.

[JLG+14] Johnson B., Laszka A., Grossklags J., Vasek M., y Moore T. (2014) Game-theoretic analysis of ddos attacks against bitcoin mining pools. In *International Conference on Financial Cryptography and Data Security*, pages 72–86. Springer.

[JS02] Japkowicz N. y Stephen S. (2002) The class imbalance problem: A systematic study. *Intelligent data analysis* 6(5): 429–449.

[JSD+17] Jordaney R., Sharad K., Dash S. K., Wang Z., Papini D., Nouretdinov I., y Cavallaro L. (2017) Transcend: Detecting concept drift in malware classification models. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 625–642.

[JSZ09] Jin Y., Sharafuddin E., y Zhang Z.-L. (2009) Unveiling core network-wide communication patterns through application traffic activity graph decomposition. *ACM SIGMETRICS Performance Evaluation Review* 37(1): 49–60.

[JWW21] Jiang J., Wang R., y Wei G.-W. (2021) Ggl-tox: Geometric graph learning for toxicity prediction. *Journal of chemical information and modeling* 61(4): 1691–1700.

[KAB+14] Kivelä M., Arenas A., Barthelemy M., Gleeson J. P., Moreno Y., y Porter M. A. (2014) Multilayer networks. *Journal of complex networks* 2(3): 203–271.

[KGVK19] Khraisat A., Gondal I., Vamplew P., y Kamruzzaman J. (2019) Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity* 2(1): 1–22.

[KKSZ11] Kriegel H.-P., Kröger P., Sander J., y Zimek A. (2011) Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1(3): 231–240.

[KW17] Kipf T. N. y Welling M. (2017) Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*.

[KZP+07] Kotsiantis S. B., Zaharakis I., Pintelas P., *et al.* (2007) Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering* 160(1): 3–24.

[LAF15] Laptev N., Amizadeh S., y Flint I. (2015) Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1939–1947.

[LF06] Leskovec J. y Faloutsos C. (2006) Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636.

[LFG+13] López V., Fernández A., García S., Palade V., y Herrera F. (2013) An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information sciences* 250: 113–141.

[LJRH11] Le D. Q., Jeong T., Roman H. E., y Hong J. W.-K. (2011) Traffic dispersion graph based anomaly detection. In *Proceedings of the Second Symposium on Information and Communication Technology*, pages 36–41.

[LL05] Leung K. y Leckie C. (2005) Unsupervised anomaly detection in network intrusion detection using clusters. In *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38*, pages 333–342.

[LLD+18] Lu J., Liu A., Dong F., Gu F., Gama J., y Zhang G. (2018) Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering* 31(12): 2346–2363.

[LMPS17] Li J., Madry A., Peebles J., y Schmidt L. (2017) Towards understanding the dynamics of generative adversarial networks. *arXiv preprint arXiv:1706.09884* .

[LPL21] Li B., Pi D., y Lin Y. (2021) Learning ladder neural networks for semi-supervised node classification in social network. *Expert Systems with Applications* 165: 113957.

[LPS21] Lombardi M., Pascale F., y Santaniello D. (2021) Internet of things: A general overview between architectures, protocols and applications. *Information* 12(2): 87.

[LTBZ15] Li Y., Tarlow D., Brockschmidt M., y Zemel R. (2015) Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493* .

[LWK+20] Long Y., Wu M., Kwoh C. K., Luo J., y Li X. (2020) Predicting human microbe-drug associations via graph convolutional network with conditional random field. *Bioinformatics* .

[MB17] Mavroeidis V. y Bromander S. (2017) Cyber threat intelligence model: an evaluation of taxonomies, sharing standards, and ontologies within cyber threat intelligence. In *2017 European Intelligence and Security Informatics Conference (EISIC)*, pages 91–98. IEEE.

[MBM+17] Monti F., Boscaini D., Masci J., Rodola E., Svoboda J., y Bronstein M. M. (2017) Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124.

[McM13] McMillan R. (2013) Definition: threat intelligence. *Gartner. https://www.gartner.com/en/documents/2487216* Accessed 01 May 2022.

[MLX+17] Mao X., Li Q., Xie H., Lau R. Y., Wang Z., y Paul Smolley S. (2017) Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802.

[MMKSM18] Madani A., Moradi M., Karargyris A., y Syeda-Mahmood T. (2018) Chest x-ray generation and data augmentation for cardiovascular abnormality classification. In *Medical Imaging 2018: Image Processing*, volumen 10574, pages 105741M. International Society for Optics and Photonics.

[MNG17] Mescheder L., Nowozin S., y Geiger A. (2017) Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. In *International Conference on Machine Learning*, pages 2391–2400. PMLR.

[MPDH19] Markopoulou D., Papakonstantinou V., y De Hert P. (2019) The new eu cybersecurity framework: The nis directive, enisa's role and the general data protection regulation. *Computer Law & Security Review* 35(6): 105336.

[MPPSD17] Metz L., Poole B., Pfau D., y Sohl-Dickstein J. (2017) Unrolled generative adversarial networks. In *5th International Conference on Learning Representations, Conference Track Proceedings*. OpenReview.net.

[MS15] Moustafa N. y Slay J. (2015) Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (MilCIS)*, pages 1–6. IEEE.

[MSS+19] Merino T., Stillwell M., Steele M., Coplan M., Patton J., Stoyanov A., y Deng L. (2019) Expansion of cyber attack data from unbalanced datasets using generative adversarial networks. In *International Conference on Software Engineering Research, Management and Applications*, pages 131–145. Springer.

[MTICGN19] Martínez Torres J., Iglesias Comesaña C., y García-Nieto P. J. (2019) Machine learning techniques applied to cybersecurity. *International Journal of Machine Learning and Cybernetics* 10(10): 2823–2836.

[NCT16] Nowozin S., Cseke B., y Tomioka R. (2016) f-gan: Training generative neural samplers using variational divergence minimization. *Advances in neural information processing systems* 29.

[NMB+18] Neal B., Mittal S., Baratin A., Tantia V., Scicluna M., Lacoste-Julien S., y Mitliagkas I. (2018) A modern take on the bias-variance tradeoff in neural networks. *arXiv preprint arXiv:1810.08591* .

[O+04] Osborne M. J. *et al.* (2004) *An introduction to game theory*, volumen 3. Oxford university press New York.

[OCWM14] Oltramari A., Cranor L. F., Walls R. J., y McDaniel P. D. (2014) Building an ontology of cyber security. In *STIDS*, pages 54–61. Citeseer.

[ODY+19] Oak R., Du M., Yan D., Takawale H., y Amit I. (2019) Malware detection on highly imbalanced data through sequence modeling. In *Proceedings of the 12th ACM Workshop on artificial intelligence and security*, pages 37–48.

[OHB19] Oh J.-H., Hong J. Y., y Baek J.-G. (2019) Oversampling method using outlier detectable generative adversarial network. *Expert Systems with Applications* 133: 1–8.

[ONJ13] Omar S., Ngadi A., y Jebur H. H. (2013) Machine learning techniques for anomaly detection: an overview. *International Journal of Computer Applications* 79(2).

[OT20] Oba T. y Taniguchi T. (2020) Graph convolutional network-based suspicious communication pair estimation for industrial control systems. *arXiv preprint arXiv:2007.10204* .

[PBS17] Pascual S., Bonafonte A., y Serrà J. (2017) Segan: Speech enhancement generative adversarial network. In *INTERSPEECH*, pages 3642–3646.

[PCSJ20] Pereira R. M., Costa Y. M., y Silla Jr C. N. (2020) Mltl: A multi-label approach for the tomek link undersampling algorithm. *Neurocomputing* 383: 95–105.

[PPJ+19] Pendlebury F., Pierazzi F., Jordaney R., Kinder J., y Cavallaro L. (2019) {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 729–746.

[PRP21] Putra I. S., Rukmono S. A., y Perdana R. S. (2021) Abstract syntax tree (ast) and control flow graph (cfg) construction of notasi algoritmik. In *2021 International Conference on Data and Software Engineering (ICoDSE)*, pages 1–6. IEEE.

[PRT+17] Pokhrel N. R., Rodrigo H., Tsokos C. P., *et al.* (2017) Cybersecurity: Time series predictive modeling of vulnerabilities of desktop operating system using linear and non-linear approach. *Journal of Information Security* 8(04): 362.

[PYT20] Pei X., Yu L., y Tian S. (2020) Amalnet: A deep learning framework based on graph convolutional networks for malware detection. *Computers & Security* 93: 101792.

[QCSSL08] Quiñonero-Candela J., Sugiyama M., Schwaighofer A., y Lawrence N. D. (2008) *Dataset shift in machine learning*. Mit Press.

[RAC+20] Rendon E., Alejo R., Castorena C., Isidro-Ortega F. J., y Granda-Gutierrez E. E. (2020) Data sampling methods to deal with the big data multi-class imbalance problem. *Applied Sciences* 10(4): 1276.

[Raf05] Rafiei D. (2005) Effectively visualizing large networks through sampling. In *VIS 05. IEEE Visualization, 2005.*, pages 375–382. IEEE.

[RHXH19] Rong Y., Huang W., Xu T., y Huang J. (2019) Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903* .

[RLNH17] Roth K., Lucchi A., Nowozin S., y Hofmann T. (2017) Stabilizing training of generative adversarial networks through regularization. *Advances in neural information processing systems* 30.

[RLWFM17] Rosca M., Lakshminarayanan B., Warde-Farley D., y Mohamed S. (2017) Variational approaches for auto-encoding generative adversarial networks. *arXiv preprint arXiv:1706.04987* .

[ROC+20] Rizvi S., Orr R., Cox A., Ashokkumar P., y Rizvi M. R. (2020) Identifying the attack surface for iot network. *Internet of Things* 9: 100162.

[Rud16] Ruder S. (2016) An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* .

[SBE19] Saad S., Briguglio W., y Elmiligi H. (2019) The curious case of machine learning in malware detection. *arXiv preprint arXiv:1905.07573* .

[SC14] Samonas S. y Coss D. (2014) The cia strikes back: Redefining confidentiality, integrity and availability in security. *Journal of Information System Security* 10(3).

[SC21] Saxena D. y Cao J. (2021) Generative adversarial networks (gans) challenges, solutions, and future directions. *ACM Computing Surveys (CSUR)* 54(3): 1–42.

[Sch14] Schall D. (2014) Link prediction in directed social networks. *Social Network Analysis and Mining* 4(1): 1–14.

[Sco20] Scornet E. (2020) Trees, forests, and impurity-based variable importance. *arXiv preprint arXiv:2001.04295* .

[SGT+08] Scarselli F., Gori M., Tsoi A. C., Hagenbuchner M., y Monfardini G. (2008) The graph neural network model. *IEEE Transactions on Neural Networks* 20(1): 61–80.

[SH18] Souri A. y Hosseini R. (2018) A state-of-the-art survey of malware detection approaches using data mining techniques. *Human-centric Computing and Information Sciences* 8(1): 1–22.

[She14] Shen L. (2014) The nist cybersecurity framework: Overview and potential impacts. *Scitech Lawyer* 10(4): 16.

[SKB+20] Sarker I. H., Kayes A., Badsha S., Alqahtani H., Watters P., y Ng A. (2020) Cybersecurity data science: an overview from machine learning perspective. *Journal of Big Data* 7(1): 1–29.

[SKK20] Shin S.-Y., Kang Y.-W., y Kim Y.-G. (2020) Android-gan: Defending against android pattern attacks using multi-modal generative network as anomaly detector. *Expert Systems with Applications* 141: 112964.

[SKPM20] Sarkar S., Khatedi N., Pramanik A., y Maiti J. (2020) An ensemble learning-based undersampling technique for handling class-imbalance problem. In *Proceedings of ICETIT 2019*, pages 586–595. Springer.

[SKR+16] Stivala A. D., Koskinen J. H., Rolls D. A., Wang P., y Robins G. L. (2016) Snowball sampling for estimating exponential random graph models for large networks. *Social Networks* 47: 167–188.

[SKWW07] Sun Y., Kamel M. S., Wong A. K., y Wang Y. (2007) Cost-sensitive boosting for classification of imbalanced data. *Pattern recognition* 40(12): 3358–3378.

[SRP12] Saini H., Rao Y. S., y Panda T. C. (2012) Cyber-crimes and their impacts: A review. *International Journal of Engineering Research and Applications* 2(2): 202–209.

[SS11] Spielman D. A. y Srivastava N. (2011) Graph sparsification by effective resistances. *SIAM Journal on Computing* 40(6): 1913–1926.

[SSMB17] Sauerwein C., Sillaber C., Mussmann A., y Breu R. (2017) Threat intelligence sharing platforms: An exploratory study of software vendors and research perspectives. *Wirtschaftsinformatik* .

[Su11] Su M.-Y. (2011) Real-time anomaly detection systems for denial-of-service attacks by weighted k-nearest-neighbor classifiers. *Expert Systems with Applications* 38(4): 3492–3498.

[SW15] Salem M. B. y Wacek C. (2015) Enabling new technologies for cyber security defense with the icas cyber security ontology. In *STIDS*, pages 42–49.

[SWL12] Singh A., Walenstein A., y Lakhotia A. (2012) Tracking concept drift in malware families. In *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, pages 81–92.

[SWM05] Stumpf M., Wiuf C., y May R. (04 2005) Subnets of scale-free networks are not scale-free: Sampling properties of networks. *Proceedings of the National Academy of Sciences of the United States of America* 102: 4221–4.

[SYWL20] Sun X., Yang J., Wang Z., y Liu H. (2020) Hgdom: Heterogeneous graph convolutional networks for malicious domain detection. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE.

[SYZ+16] Song G., Ye Y., Zhang H., Xu X., Lau R. Y., y Liu F. (2016) Dynamic clustering forest: an ensemble framework to efficiently classify textual data stream with concept drift. *Information Sciences* 357: 125–143.

[SZ12] Silva T. C. y Zhao L. (2012) Network-based high level data classification. *IEEE Transactions on Neural Networks and Learning Systems* 23(6): 954–970.

[SZ16] Silva T. C. y Zhao L. (2016) *Machine learning in complex networks*, volumen 1. Springer.

[SZW+19] Shamsolmoali P., Zareapoor M., Wang R., Jain D. K., y Yang J. (2019) G-ganisr: Gradual generative adversarial network for image super resolution. *Neurocomputing* 366: 140–153.

[TBS+18] Toch E., Bettini C., Shmueli E., Radaelli L., Lanzi A., Riboni D., y Lepri B. (2018) The privacy implications of cyber security systems: A technological survey. *ACM Computing Surveys (CSUR)* 51(2): 1–27.

[TK11] Thang T. M. y Kim J. (2011) The anomaly detection by using dbscan clustering with multiple parameters. In *2011 International Conference on Information Science and Applications*, pages 1–5. IEEE.

[TR18] Tounsi W. y Rais H. (2018) A survey on technical threat intelligence in the age of sophisticated cyber attacks. *Computers & security* 72: 212–233.

[Tsy04] Tsymbal A. (2004) The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin* 106(2): 58.

[UAB19] Ucci D., Aniello L., y Baldoni R. (2019) Survey of machine learning techniques for malware analysis. *Computers & Security* 81: 123–147.

[UQR+19] Usama M., Qadir J., Raza A., Arif H., Yau K.-L. A., Elkhatib Y., Hussain A., y Al-Fuqaha A. (2019) Unsupervised machine learning for networking: Techniques, applications and research challenges. *IEEE access* 7: 65579–65615.

[VEH20] Van Engelen J. E. y Hoos H. H. (2020) A survey on semi-supervised learning. *Machine Learning* 109(2): 373–440.

[VNVL21] Vo M. T., Nguyen T., Vo H. A., y Le T. (2021) Noise-adaptive synthetic oversampling technique. *Applied Intelligence* pages 1–10.

[VPT16] Vondrick C., Pirsiavash H., y Torralba A. (2016) Generating videos with scene dynamics. *Advances in neural information processing systems* 29.

[VTM14] Vasek M., Thornton M., y Moore T. (2014) Empirical analysis of denial-of-service attacks in the bitcoin ecosystem. In *International conference on financial cryptography and data security*, pages 57–71. Springer.

[VWK+20] Verbraeken J., Wolting M., Katzy J., Kloppenburg J., Verbelen T., y Rellermeyer J. S. (2020) A survey on distributed machine learning. *ACM Computing Surveys (CSUR)* 53(2): 1–33.

[WC09] Wasikowski M. y Chen X.-w. (2009) Combating the small sample class imbalance problem using feature selection. *IEEE Transactions on knowledge and data engineering* 22(10): 1388–1400.

[WCA+16] Wu Y., Cao N., Archambault D., Shen Q., Qu H., y Cui W. (2016) Evaluation of graph sampling: A visualization perspective. *IEEE transactions on visualization and computer graphics* 23(1): 401–410.

[WFZ16] Wehmuth K., Fleury É., y Ziviani A. (2016) On multiaspect graphs. *Theoretical Computer Science* 651: 50–61.

[WK96] Widmer G. y Kubat M. (1996) Learning in the presence of concept drift and hidden contexts. *Machine learning* 23(1): 69–101.

[WLC+13] Wei W., Li J., Cao L., Ou Y., y Chen J. (2013) Effective detection of sophisticated online banking fraud on extremely imbalanced data. *World Wide Web* 16(4): 449–475.

[WLY+20] Wang P., Li S., Ye F., Wang Z., y Zhang M. (2020) Packetcgan: Exploratory study of class imbalance for encrypted traffic classification using cgan. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE.

[WPC+20] Wu Z., Pan S., Chen F., Long G., Zhang C., y Philip S. Y. (2020) A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* .

[WSH+18] Wu L., Sun P., Hong R., Fu Y., Wang X., y Wang M. (2018) Socialgcn: An efficient graph convolutional network based model for social recommendation. *arXiv preprint arXiv:1811.02815* .

[WSH+20] Wang W., Shang Y., He Y., Li Y., y Liu J. (2020) Botmark: Automated botnet detection with hybrid analysis of flow-based and graph-based traffic behaviors. *Information Sciences* 511: 284–296.

[WSW21] Wang Z., She Q., y Ward T. E. (2021) Generative adversarial networks in computer vision: A survey and taxonomy. *ACM Computing Surveys (CSUR)* 54(2): 1–38.

[WYA19] Wang X., Yang I., y Ahn S.-H. (2019) Sample efficient home power anomaly detection in real time using semi-supervised learning. *IEEE Access* 7: 139712–139725.

[WZ07] Wang F. y Zhang C. (2007) Label propagation through linear neighborhoods. *IEEE Transactions on Knowledge and Data Engineering* 20(1): 55–67.

[WZF15] Wehmuth K., Ziviani A., y Fleury E. (2015) A unifying model for representing time-varying graphs. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10. IEEE.

[XG18] Xie T. y Grossman J. C. (2018) Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Physical review letters* 120(14): 145301.

[XKL+18] Xin Y., Kong L., Liu Z., Chen Y., Li Y., Zhu H., Gao M., Hou H., y Wang C. (2018) Machine learning and deep learning methods for cybersecurity. *Ieee access* 6: 35365–35381.

[XSY+21] Xia F., Sun K., Yu S., Aziz A., Wan L., Pan S., y Liu H. (2021) Graph learning: A survey. *IEEE Transactions on Artificial Intelligence* 2(2): 109–127.

[XWMZ17] Xu D., Wang Y., Meng Y., y Zhang Z. (2017) An improved data anomaly detection method based on isolation forest. In *2017 10th international symposium on computational intelligence and design (ISCID)*, volumen 2, pages 287–291. IEEE.

[XZLH20] Xu L., Zeng X., Li W., y Huang Z. (2020) Multi-granularity generative adversarial nets with reconstructive sampling for image inpainting. *Neurocomputing* .

[YCWX18] Yang Z., Chen W., Wang F., y Xu B. (2018) Generative adversarial training for neural machine translation. *Neurocomputing* 321: 146–155.

[YHZ+21] Yu Z., Huang F., Zhao X., Xiao W., y Zhang W. (2021) Predicting drug–disease associations through layer attention graph convolutional network. *Briefings in Bioinformatics* 22(4): bbaa243.

[YLY+18] You J., Liu B., Ying Z., Pande V., y Leskovec J. (2018) Graph convolutional policy network for goal-directed molecular graph generation. In *Advances in neural information processing systems*.

[YM19] Yilmaz I. y Masum R. (2019) Expansion of cyber attack data from unbalanced datasets using generative techniques. *arXiv preprint arXiv:1912.04549* .

[YYJ19] Yan J., Yan G., y Jin D. (2019) Classifying malware represented as control flow graphs using deep graph convolutional neural network. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 52–63. IEEE.

[ZBB+20] Zola F., Bruse J. L., Barrio X. E., Galar M., y Urrutia R. O. (2020) Generative adversarial networks for bitcoin data augmentation. In *2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, pages 136–143. IEEE.

[ZCFM06] Zhang S., Chakrabarti A., Ford J., y Makedon F. (2006) Attack detection in time series for recommender systems. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 809–814.

[ZCZ+18] Zhou J., Cui G., Zhang Z., Yang C., Liu Z., Wang L., Li C., y Sun M. (2018) Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* .

[ZCZ20] Zhang Z., Cui P., y Zhu W. (2020) Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering* .

[ZEBU19] Zola F., Eguimendia M., Bruse J. L., y Urrutia R. O. (2019) Cascading machine learning to attack bitcoin anonymity. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 10–17. IEEE.

[ZLL+19] Zheng L., Li Z., Li J., Li Z., y Gao J. (2019) Addgraph: Anomaly detection in dynamic graph using attention-based temporal gcn. In *IJCAI*, pages 4419–4425.

[ZLN+20] Zhao T., Liu Y., Neves L., Woodford O., Jiang M., y Shah N. (2020) Data augmentation for graph neural networks. *arXiv preprint arXiv:2006.06830* .

[ZLY+20] Zhao J., Liu X., Yan Q., Li B., Shao M., y Peng H. (2020) Multi-attributed heterogeneous graph convolutional network for bot detection. *Information Sciences* 537: 380–393.

[ZP17] Zhou C. y Paffenroth R. C. (2017) Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 665–674.

[ZPSZ+19] Zola F., Pérez-Solà C., Zubia J. E., Eguimendia M., y Herrera-Joancomartí J. (2019) Kriptosare. gen, a dockerized bitcoin testbed: analysis of server performance. In *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5. IEEE.

[ZTXM19] Zhang S., Tong H., Xu J., y Maciejewski R. (2019) Graph convolutional networks: a comprehensive review. *Computational Social Networks* 6(1): 1–23.

[ZYL+20] Zhao J., Yan Q., Liu X., Li B., y Zuo G. (2020) Cyber threat intelligence modeling based on heterogeneous graph convolutional network. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2020)*, pages 241–256.

[ZZ20] Zheng W. y Zhao H. (2020) Cost-sensitive hierarchical classification for imbalance classes. *Applied Intelligence* 50(8): 2328–2338.

Part II. Published, accepted and submitted publications

1. *Bitcoin and cybersecurity: Temporal dissection of blockchain data to unveil changes in entity behavioral patterns*

The work related to this part is:

- Zola F., Bruse J.L., Eguimendia M., Galar M. and Orduna Urrutia R., *Bitcoin and cybersecurity: Temporal dissection of blockchain data to unveil changes in entity behavioral patterns*, Applied Sciences, 9(23), pp.5003, DOI: 10.3390/app9235003

    - Status: Published.
    - Impact Factor (JCR 2019): 2.474.
    - Knowledge area:
        - Engineering, Multidisciplinary. Ranking 32/91 (Q2).
        - Chemistry, Multidisciplinary. Ranking 88/177 (Q2).
        - Materials Science, Multidisciplinary. Ranking 161/314 (Q3).
        - Physics, Applied. Ranking 63/155 (Q2).

applied sciences

MDPI

Article

Bitcoin and Cybersecurity: Temporal Dissection of Blockchain Data to Unveil Changes in Entity Behavioral Patterns

Francesco Zola [1,*], Jan Lukas Bruse [1], Maria Eguimendia [1], Mikel Galar [2] and Raul Orduna Urrutia [1]

[1] Vicomtech, 20009 Donostia/San Sebastian, Spain; jbruse@vicomtech.org (J.L.B.); meguimendia@vicomtech.org (M.E.); rorduna@vicomtech.org (R.O.U.)
[2] Institute of Smart Cities, Public University of Navarre, 31006 Pamplona, Spain; mikel.galar@unavarra.es
* Correspondence: fzola@vicomtech.org

Received: 18 October 2019; Accepted: 16 November 2019; Published: 20 November 2019

Abstract: The Bitcoin network not only is vulnerable to cyber-attacks but currently represents the most frequently used cryptocurrency for concealing illicit activities. Typically, Bitcoin activity is monitored by decreasing anonymity of its entities using machine learning-based techniques, which consider the whole blockchain. This entails two issues: first, it increases the complexity of the analysis requiring higher efforts and, second, it may hide network micro-dynamics important for detecting short-term changes in entity behavioral patterns. The aim of this paper is to address both issues by performing a "temporal dissection" of the Bitcoin blockchain, i.e., dividing it into smaller temporal batches to achieve entity classification. The idea is that a machine learning model trained on a certain time-interval (batch) should achieve good classification performance when tested on another batch if entity behavioral patterns are similar. We apply cascading machine learning principles—a type of ensemble learning applying stacking techniques—introducing a "k-fold cross-testing" concept across batches of varying size. Results show that blockchain batch size used for entity classification could be reduced for certain classes (Exchange, Gambling, and eWallet) as classification rates did not vary significantly with batch size; suggesting that behavioral patterns did not change significantly over time. Mixer and Market class detection, however, can be negatively affected. A deeper analysis of Mining Pool behavior showed that models trained on recent data perform better than models trained on older data, suggesting that "typical" Mining Pool behavior may be represented better by recent data. This work provides a first step towards uncovering entity behavioral changes via temporal dissection of blockchain data.

Keywords: Bitcoin analysis; behavioral patterns; machine learning; time-series analysis; entities detection; ensemble learning

1. Introduction

Bitcoin is a decentralized peer-to-peer cryptocurrency (or crypto) where all transactions are stored in a blockchain [1]—a public ledger that cannot be manipulated or changed [2]. Its features have made Bitcoin one of the most frequently used and priced cryptocurrencies.

The increasing price of Bitcoin has recently raised questions concerning the cybersecurity of the Bitcoin network. Its blockchain architecture makes Bitcoin almost invulnerable and too hard to attack [3], which has led hackers to directly attack single nodes (entities) of the network.

According to a study presented in [4], entities such as Exchanges, Mining Pools, Gambling operators, eWallets, and financial services are much more likely to be attacked than other services.

In particular, Ref. [4] demonstrated that the size of Mining Pools, for example, is related to the probability of being a target of an attack.

Bitcoin not only is vulnerable to cyber-attacks but is also the most frequently used cryptocurrency for concealing illicit activities, as confirmed by Jonathan Levin, co-founder and COO of Chainalysis [5,6]. In [7], authors state that Bitcoin is the dominant cryptocurrency used in criminal activities due to the non-transparent transactions and due to the lack of effective regulatory mechanisms. Fanusie et al. [8] describes that, from 2013 to 2016, the number of illicit entities (Market, Ponzi, Malware, etc.) were multiplied by five and two types of services—Mixers and Gambling—increased their volume of transactions involved in money laundering activities.

Hence, a crucial step to address Bitcoin network security and detect both cyber-attacks and illicit activities is to uncover the network behavior of sensible targets—of entities potentially involved in or targets of illicit activities. Generally, this is achieved by reducing the anonymity of users. Although Bitcoin is usually described as an anonymous system, it is actually pseudo-anonymous [9], as it is possible to track, identify, and classify Bitcoin entities within the blockchain network by joining public addresses and private keys.

Prior studies have tried to classify entities according to classes representing specific entity behavior within the network [10–12]. These techniques usually consider the whole blockchain and thus classification is performed considering all network (macro) dynamics among users. First of all, using the entire blockchain data typically comes at high computational and resource costs, which could be reduced by considering smaller amounts of input data. Furthermore, increasing the "temporal resolution" by considering shorter time intervals of blockchain data may highlight network micro-dynamics, i.e., small, short-term behavioral changes that can change the classification of an entity over time. These changes in classification could be useful for detecting suspicious activities. For example, a usually well-performing model for a certain entity class that all of a sudden fails to detect known elements of that class could be a hint for critical behavioral changes related to attacks or illicit activities.

The key idea behind our study is that a model trained on a batch with a certain time-interval should achieve good classification performance when testing it on another temporal batch—if similarity between behavioral patterns of the detected entities within those batches is sufficiently high.

It is to be noted that the aim of this work is not to directly investigate cyber-attacks or to improve classification rates of Bitcoin entities, but to investigate the effect of considering Bitcoin blockchain data as a time-series, dividing it into temporal batches and studying micro-changes of behavioral entity patterns over time. Results could highlight which entities show relatively consistent behavior from batch to batch (making them less likely to be affected by illicit activities and narrowing down the time interval necessary to obtain "typical" entity behavior) and which entities show strongly varying behavior over time (thus revealing short-term changes in behavior that could suggest suspicious network activities or sensible targets).

We believe that our novel approach could be valuable for future forensic tools as it may highlight the importance of considering blockchain data as a time-series to reveal short-term behavioral changes, hence shifting the paradigm of forensic blockchain analysis from a "macro" or "black-box" approach towards taking into account network micro-dynamics. As a byproduct, these considerations can reduce the complexity and the required resources for forensics analysis.

Specifically, the present analysis is based on extending cascading machine learning concepts presented in Zola et al. [10], which essentially apply ensemble learning and stacking of several classifiers trained with different datasets derived from blockchain data. The idea behind this technique is to classify Bitcoin entities using prior classification information, combine them with new blockchain-derived data, and then compute a final classification to improve predictive performance.

All data were normalized with respect to the considered batches (time-intervals). This operation allows us to generalize the derived classifiers and use them with data from different sources or from other blockchains, which increases transferability of our results.

We carry out four experiments considering different batch sizes, where a cascading approach with data extracted from blockchain batches is used. We started using a batch size of 60,000 consecutive blocks (approx. 12 months of data); in the second experiment we used 30,000 consecutive blocks (approx. six months of data); in the third experiment we chose 20,000 consecutive blocks (approx. four months of data), and in the last experiment we used 10,000 consecutive blocks (approx. two months of data). In each experiment, we used one batch to train the initial classifiers, meanwhile the others—left out from training—were used to test them. This procedure is repeated for each batch extracted from the blockchain until each one has been used once for training ("k-fold cross-testing"). These tests allow us to analyze similarities between entity behavior over time and investigate if certain behavioral patterns are repeated across different blockchain batches. We consider six different types of Bitcoin entities: Exchange, Gambling, Marketplace, Mining Pool, Mixer, and eWallet. At the end of the four experiments, following previous studies [4,13] that considered Mining Pool entities as sensitive targets (potentially subject to cyber-attacks), a deeper analysis of Mining Pool behavior is shown.

The rest of the paper is organized as follows. Section 2 describes related work. Afterwards, Section 3 presents an overview of the used data. Section 4 introduces the graph models and the machine learning models implemented; then, in Section 5, the experimental analysis is presented. Finally, in Sections 6 and 7, we draw conclusions and provide some guidelines for future work.

2. Related Works

Bitcoin properties have led hackers to deploy various cyber-attacks in order to introduce chaos/noise into the network and take advantage of it. In [14], an analysis about the impact of Distributed Denial-of-Service (DDoS) attacks on the volume traded on the Exchange is presented, in particular regarding 17 attack cases that occurred across 2016 and 2018. In [15], transactions related to 35 ransomware families (malware) are analyzed, determining a minimum market worth about USD 12,768,536. These kinds of attacks allowed hackers to steal Bitcoin (BTC); for example, as admitted by Changpeng Zhao (CEO of Binance, one of the largest Exchanges), hackers stole more than 7000 BTC [16], about 40 USD million in 7 May 2019, considering the exchange rate (closing market value on that day) equals USD 5829.50 [17].

Criminal concerns related to the Bitcoin network can be grouped into three classes: Bitcoin-specific crimes, money laundering and Bitcoin-facilitated crimes [18]. User anonymity not only has been the keystone for dissemination of cryptocurrencies, but it has promoted illegal activities and cyber-attacks within the Bitcoin network. In fact, as discussed in the Introduction, knowing the identity of users involved in the network helps determine sensible targets [4]. Moreover, short-term changes of entity behavior, for example in Mining Pools [13], could be symptoms of a cyber-attack.

In [19,20], current measures used by the Bitcoin protocol to preserve anonymity within the network are analyzed. As shown in [21], such measures are, however, not enough to protect user privacy. In fact, it is possible to decrease Bitcoin network anonymity by using address clustering and combining information from various sources. For example, in [22], an address clustering is computed in order to identify the CryptoLocker (a family of ransomware) and in [23] the clustering is based on conservative constraints (patterns). Reid et al. [24] exploit topological and external information in order to investigate a large theft of Bitcoins. Meanwhile, Fleder et al. make use of information scraped from forums and social media in order to characterize known and unknown users [25].

In [26], a study on two graphs generated by the Bitcoin transaction network using anomaly detection techniques is presented, aiming to detect which users and transactions were the most suspicious. Nevertheless, Monamo et al. [27] uses an unsupervised learning algorithm for classifying anomalies (financial fraud and money laundering) on the Bitcoin network based on transaction patterns. An unsupervised k-means classifier is applied in [28] in order to identify atypical transactions related to money laundering. Bartoletti et al. [29] use machine learning algorithms to identify Ponzi schemes in the Bitcoin network. Yin et al. in [30] apply supervised learning techniques in order to determine a "big picture" of cybercrime-related entities in the Bitcoin ecosystem.

In [10,12], methods for attacking Bitcoin user anonymity are presented. Both methods use the whole blockchain to create supervised machine learning models and classify Bitcoin entities. In particular, a cascading machine learning model is introduced in [10], which is essentially ensemble learning based on the stacking concept presented in [31]. The idea of the cascading model is to implement a cascade of (weak) classifiers, such that prior classification results can be joined and can be used to enrich a final (strong) classification. The cascading machine learning approach is compared with other techniques as [11,32], and showed that the new approach not only reduces the complexity of the model by reducing the features implied in the classification, but also reaches a very high accuracy considering six Bitcoin entity classes.

In anomaly detection, changes in network traffic are often a symptom of attacks. Nevertheless, it is generally difficult to detect and evaluate these changes in early stages of an attack as such changes in traffic cannot easily be distinguished from usual traffic fluctuation [33]. The following studies applied division of data into subsets or batches, similar to the work presented in this paper. In [34], a new process for training a hidden Markov model (HMM) to detect a denial-of-service attack (DoS) in program behavior data (system calls produced by processes of a program) is described, aiming to reduce training times. Authors first divide the long observation sequence into multiple subsets of sequences. Then, they join all the generated sub-models reducing the training time by about 60% compared to conventional training.

The approach of dividing information in subsets is used as well when analyzing graph structures. In these cases, the evolution of the graph over time is analyzed, creating a graph stream [35]. For example, in [36], a monthly call graph is divided into weekly snapshots in order to consider new dynamics of the call network and achieve churn prediction. Furthermore, in [37], a parallel partitioning approach to discover cyber-threats in computer network traffic focusing on substructures is presented.

In this paper, the idea is to join the machine learning model introduced in [10] with the concept of batch analysis applied, for example, in anomalies detection [34,38]. We aim to add a "temporal view" to Bitcoin entity analysis in order to unveil changes in behavioral patterns over time. This approach provides a magnifying lens allowing us (a) to study how using smaller batches of Bitcoin data affects classification performance and (b) to analyze micro-dynamics present in the Bitcoin blockchain network. All used features are normalized in order to reduce dependency on the chosen block size, and results may help data scientists reduce the complexity and the size of the initial dataset in forensic analysis.

3. Datasets

In studies of Bitcoin behavior, it is difficult to identify a ground-truth labeled dataset due to the anonymity of the network and due to the activity of its entities, which can change over time. However, here we aim to analyze these changes and determine how they affect the entity behavior classification. Such change in the classification over time may help us determine suspicious situations and sensitive targets.

We therefore used two datasets, the first one downloaded from WalletExplorer [39]—a platform that can be considered a benchmark for Bitcoin entities detection—and the second one consisting of Bitcoin blockchain data downloaded from the mainnet [40].

WalletExplorer is a web-page where data and information about different known entities detected until today are collected. The dataset is continuously being updated and its information has been used as a starting point for many Bitcoin-related studies, such as [15,32,41]. For our purpose, each entity was downloaded with its origin name and its related detected addresses.

The downloaded entities belong to seven different classes:

- *Exchange*: entities that allow their customers to trade cryptocurrencies or to exchange cryptos for fiat currencies (or vice versa);
- *Gambling*: entities that offer gambling services based on Bitcoin currency (casino, betting, roulette, etc.);

- *Mining Pool*: entities composed of a group of miners that work together sharing their resources in order to reduce the volatility of their returns;
- *Mixer*: entities that offer a service to obscure the traceability of their Bitcoin clients' transactions;
- *Marketplace*: entities allowing to buy any kind of goods or services using cryptocurrencies. They are frequently used to buy illegal goods;
- *eWallet*: entities that allow an individual to create online accounts that can be used to receive and send money. Users never need to download the Bitcoin software themselves and all of the user's transactions are made on behalf of the user by the eWallet service, using keys controlled by the service [23];
- *Lending*: entities that allow users to lend Bitcoins and passively earn interests on it, or allow them to request a loan.

As shown in Table 1, 315 different entities and more than 18,000,000 addresses were downloaded from WalletExplorer. However, after creating a first overview of the dataset, we decided to not use the *Lending* class because seven entities were considered too few to implement and train the cascading machine learning system.

Table 1. Overview of WalletExplorer entities and address data.

Class	Abbrev.	# Entity Total	# Entity 3 years	% Entity Ratio	# Address Total	# Address 3 years	% Address Ratio
Exchange	Ex	137	124	90.51	9,950,742	6,361,096	63.93
Gambling	Gmb	76	59	77.63	3,054,477	1,711,407	56.03
Marketplace	Mrk	20	19	95.00	2,349,300	171,966	7.32
Mining Pool	Pool	25	17	68.00	76,297	43,041	56.41
Mixer	Mxr	37	35	94.59	476,400	273,228	57.35
eWallet	eWal	13	13	100.0	2,604,111	2,191,129	84.14
Lending	Len	7	-	-	113,900	-	-
Total		**315**	**267**		**18,625,227**	**10,751,867**	

The second dataset was directly downloaded from the Bitcoin mainnet through the Bitcoin Core program [42]. Our analysis focuses on the last (about) three years of lifetime of Bitcoin blockchain only, so we used the Bitcoin blockchain data created from blocks height 390,000 to 570,000, corresponding to blocks mined from 24 December 2015, 5:33:51 p.m. until 3 April 2019, 9:20:08 a.m., respectively. This decision was taken in order to decrease the computational cost for carrying out the experiments.

Table 1 shows the amount of different entities and addresses calculated in the last three years as well, and the ratio between the samples in the considered time interval compared to the whole population.

The final dataset was composed by data belonging to six classes: *Exchange*, *Gambling*, *Marketplace*, *Mining Pool*, *Mixer*, and *eWallet*. All entities were represented by more than 65% of distinct samples, and also by more than 50% of distinct addresses, except for the *Marketplace* with just 7.32% of samples. A first analysis highlighted that these *Marketplace* entities were mainly active in the first six years of the blockchain lifetime. Then, over time, they were closed from Law Enforcement Officers (LEOs) in case of illicit activities or owners decided to close their services.

In this study, the downloaded Bitcoin data of the last three years were cross-referenced with the (labelled) WalletExplorer data in order to re-size the original dataset and remove all unlabelled and unusable data for our supervised cascading machine learning approach.

4. Methodology

In this section, the four graph models used in this work are presented: address-transaction graph, entity-transaction graph, and 1_motif and 2_motif graph. Each graph representation allows us to analyze a different aspect of the behavior of an entity. These graph models represent the sources for

extracting dataframes (tabulated data), which were used for implementing the cascading machine learning models.

4.1. Blockchain and Motifs' Graphs

Blockchain data can be directly represented through an address-transaction graph (Figure 1). In this graph, vertices represent addresses and transactions, while directed edges (arrows) between addresses and transactions indicate incoming relations, and directed edges between transactions and addresses correspond to outgoing relations. Each directed edge can also include additional information such as values, time-stamps, etc.

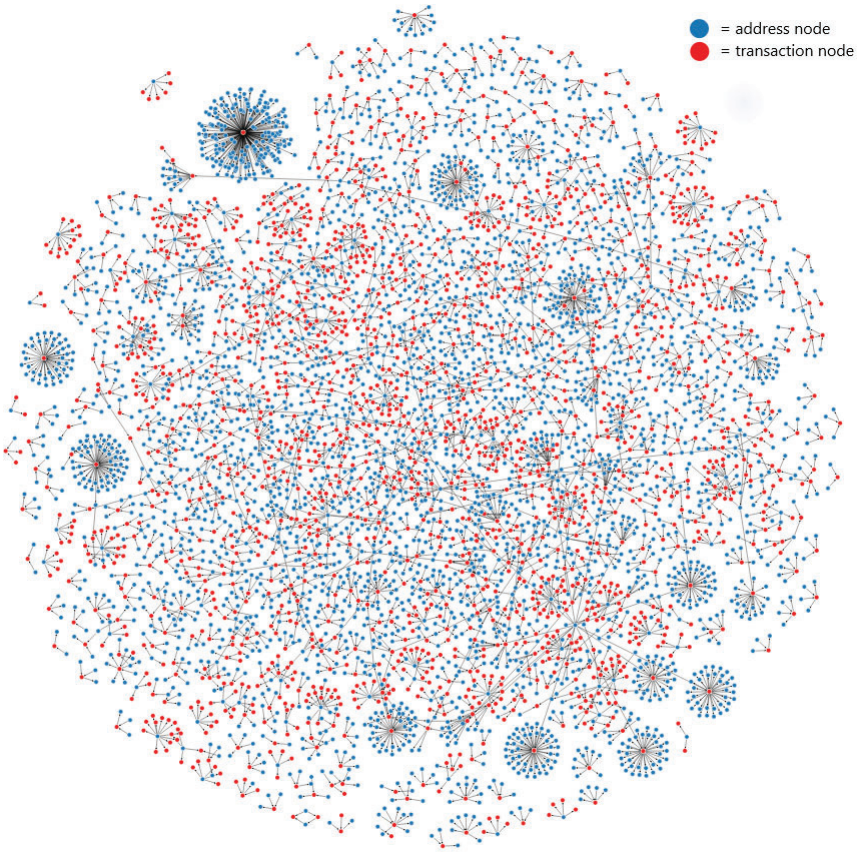


Figure 1. Address-transaction graph computed with one block of the Bitcoin mainnet.

According to Bitcoin.org [43], each single user has the responsibility for adopting good practices in order to protect his/her anonymity and remain private. The basic recommendation is to use a new Bitcoin address for any new payment, and additionally use multiple wallets for different purposes. These suggestions create new dynamics in the network increasing the complexity of user behavior detection. Nevertheless, over time, these same concepts were used to define several heuristic properties with the aim to find related addresses for subsequent address clustering. This process is used to find all addresses that belong to a certain user, and this allows us to introduce the *entity* concept. In particular, we refer to an entity as a physical person or organization related to one or multiple public key addresses belonging to one or more wallets. Using these clusters of addresses and the relation among transactions, it is possible to create the entity-transaction graph starting from the address-transaction graph.

Heuristic properties have been developed and presented in different studies such as [20,23,44]. However, in this study, we do not need to apply heuristic assumptions since, as indicated in Section 3, the Bitcoin blockchain data were combined with labelled data gathered from WalletExplorer, such that relations among addresses (i.e., clusters) are known.

The motifs graph used here is usually applied in bioinformatics, specifically in metabolic network analysis [45]. However, prior studies such as [11,32] have successfully translated the concept of motifs to Bitcoin analyses.

According to [10], a N_motif graph is a graph composed by paths from the entity-transaction graph with length $2N$ that starts and ends with an entity. Let $(e_1, .., e_M) \in E$ be a class of entities and $(t_1, .., t_N) \in T$ be a class of transactions, with $M \leq N + 1$, then:

$$N\_motif = (e_1, t_1, ..., t_N, e_M). \quad (1)$$

From this particular graph, it is possible to extract information concerning the relations among entities, but also concerning the topology that such relations create among entities. If two different entities are connected through one transaction, the topology is called Direct Distinct. If one entity is connected with itself (again through one transaction) the topology is called Direct Loop (Figure 2).

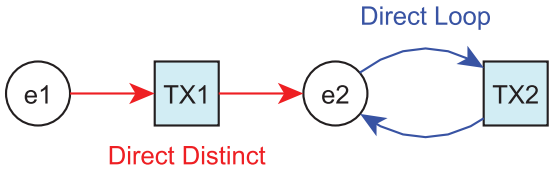


Figure 2. Example of 2_motif graph topology.

Starting from the results presented in [10], in the present study, only the 1_motif and 2_motif graphs were used.

4.2. Temporal Batch

Larger time intervals may cover specific behavioral changes of an entity affecting its detection. Moreover, the amount of data in a larger time interval increases the complexity of the problem, often leading to the creation of highly non-transparent "black-box" models. Our approach involves dissecting data temporally, so dividing everything into smaller batches that can then be studied by themselves (thus using "smaller black-boxes", which may help to understand better the internal dynamics within the large black-box of the full data).

The idea behind our approach is that a model trained on a batch with a certain time-interval can be used for unveiling behavioral entity patterns when testing it on another temporal batch. Thereby, higher classification performance would relate to higher similarity between behavioral entity patterns within the training and test batch. Hence, in this analysis, it is important to divide the whole Bitcoin dataset into temporal batches, where each batch represents a dataset composed by the transactions belonging to a fixed number of consecutive blocks.

Let min_blk be the minimum block height and max_blk the maximum block height in the (considered) Bitcoin blockchain data, and let $batch\_size$ be the size (in blocks) of a single batch, then the number of generated batches can be computed through Equation (2)

$$\#batches = \frac{max\_blk - min\_blk}{batch\_size}. \quad (2)$$

In our analysis, we considered min_blk with a value of 390,000 blocks and max_blk with a value of 570,000, and four different values for $batch\_size$. We started creating batches of 60,000 consecutive blocks (representing approx. 12 months) in the first experiment. In the second one, we chose a value of 30,000 (representing approx. six months), in the third one, a value of 20,000 (representing approx. four months) was used and, finally, in the last experiment, the $batch\_size$ was fixed to 10,000 consecutive blocks (representing approx. two months).

Table 2 shows the number of batches considered in each experiment and the related distinct known entities divided per class. This table shows that the majority of the detected entities belonged to batches with "old or past" data, since the number of distinct known entities for almost all the classes decreases over time.

Table 2. Distinct known entities in each experiment and for each batch.

batch_size = 60,000						
# batch	# Ex	# Gmb	# Mrk	# Pool	# Mxr	# eWal
1	122	51	18	14	30	12
2	104	46	12	12	12	11
3	81	27	9	9	1	10
batch_size = 30,000						
# batch	# Ex	# Gmb	# Mrk	# Pool	# Mxr	# eWal
1	121	50	14	13	18	12
2	109	40	17	11	20	12
3	102	46	12	11	11	10
4	89	27	6	7	3	11
5	74	23	5	6	1	9
6	73	22	7	6	1	8
batch_size = 20,000						
# batch	# Ex	# Gmb	# Mrk	# Pool	# Mxr	# eWal
1	119	50	14	13	13	11
2	110	41	15	10	18	12
3	104	39	10	11	11	10
4	100	42	12	10	11	10
5	90	39	10	8	3	10
6	79	24	4	6	2	11
7	74	21	1	6	1	9
8	63	22	6	5	1	7
9	68	19	5	4	1	7
batch_size = 10,000						
# batch	# Ex	# Gmb	# Mrk	# Pool	# Mxr	# eWal
1	117	48	13	13	8	11
2	105	44	13	11	8	11
3	106	40	9	10	13	12
4	98	37	15	10	11	12
5	93	37	6	8	8	10
6	99	36	10	10	8	10
7	95	36	11	8	8	10
8	93	38	9	10	7	10
9	86	39	9	7	2	10
10	80	25	4	5	3	8
11	75	22	4	6	2	11
12	68	21	4	3	1	7
13	67	19	0	6	1	8
14	63	19	1	4	1	9
15	58	20	4	3	1	6
16	59	19	2	5	1	7
17	57	17	5	3	1	6
18	62	17	0	3	1	7

4.3. Features

In each experiment and for each temporal batch generated, the four graph models presented in Section 4.1 were created. Then, from each graph model, four dataframes (two-dimensional labelled data structure or data table with samples as rows and extracted features as columns) were created extracting several features. All the features are presented in detail in [10]:

- *Entity dataframe* contains all features extracted from the entity-transaction graph and contains a total of seven features.
- *Address dataframe* contains all features extracted from the address-transaction graph and contains a total of seven features.
- *1_motif dataframe* contains the information directly extracted from the 1_$motif$ graph and contains a total of nine features.
- *2_motif dataframe* contains information gathered from the 2_$motif$ graph and contains a total of 18 features.

In order to compare information from different batches, and in order to reduce the complexity of the models, the features (excluding the ones related to an amount or balance) were normalized on a range between 0 and 1. This normalization was computed with respect to the minimum and maximum values in the corresponding batch. This process allow us to re-use these models with other batch sizes and to use them with input data from other sources/blockchains in future studies.

Let X be the value to be normalized, $X_{\min}$ the minimum and $X_{\max}$ the maximum of the considered feature in a certain batch, then the normalized value X_{norm} was computed using Equation (3):

$$X_{\text{norm}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}. \tag{3}$$

4.4. Cascading Machine Learning

In this study, we used and extended the cascading machine learning concepts introduced in [10], which have shown excellent classification results when using the whole blockchain data. As mentioned in Section 2, the cascading machine learning optimizes the number of features used in the classification and ensures high accuracy value, comparing them with output accuracy of other techniques [10]. Our cascading model is a type of ensemble learning applying stacking techniques [46]. The idea transferred to the Bitcoin network is based on enriching Bitcoin entity data with information obtained from prior classifications creating cascading classifiers, as shown in Figure 3.

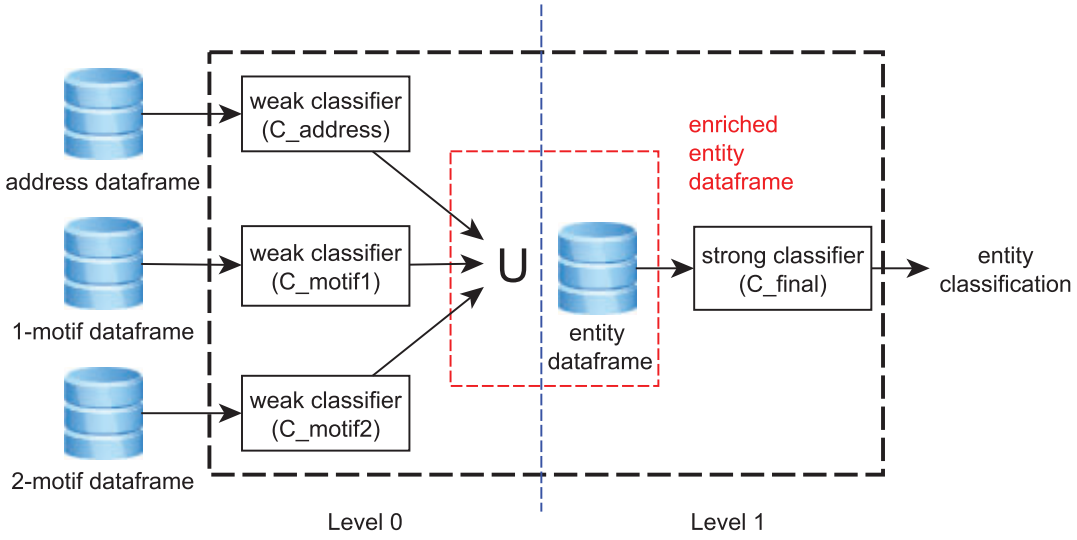


Figure 3. Cascading machine learning architecture.

The first step to create the cascading system was to split the address, 1_motif and 2_motif dataframes into train/test datasets. In particular, this operation was computed using a proportion of 50/50. Then, the created train datasets were used to implement the weak classifiers called *C_address*, *C_motif1* and *C_motif2* in relation with their own input data (zero-level in Figure 3). This process was computed by applying a k-fold cross-validation with k = 5. In k-fold cross-validation, the dataset is

divided into k folds, where $k-1$ folds are used for training the model and the one left out during the training phase is used to validate it. The whole process is repeated until each fold is used only once for validating.

Once the cross-validation was ended and the models were generated/trained, the test dataset was used as input. The output of the classification from the Level 0 classifiers was joined with the information in the entity dataframe in order to create the enriched entity dataframe. Then, this new enriched dataframe was used for training the final (strong) classifier called *C_final* (first-level in Figure 3).

Thanks to the initial dataset split into train and test, the information generated from the weak classifiers was created from a completely unseen dataset. In fact, if the data that were used to train the zero-level learner were also used to generate the enriched dataset for training the first-level learner, there would be a high risk of overfitting [47], which hence has been avoided here.

The outgoing information from the zero-level classifiers was processed following an enrichment process, as indicated in [10]. This process consists of assigning one of the six possible output classes to each entry in the input dataframe and joining the input label (from WalletExplorer) with the computed output class. Then, the input label was grouped counting how many times a sample belonging to a particular entity has been detected in each of the considered classes. This value was then normalized as indicated in Equation (4), where E represents the entities set and N is the number of considered classes ($N = 6$ in this study). The term $\| P_{e|\,\mathrm{j}} \|$ represents how many times a sample originally labelled with entity e generates a prediction belonging to class j, while the term $\sum_{i=1}^{N} \| P_{e|\,\mathrm{i}} \|$ counts all the predictions generated from samples with labelled input belonging to entity e:

$$\forall e \in E \qquad \frac{\| P_{e|\,\mathrm{j}} \|}{\sum_{i=1}^{N} \| P_{e|\,\mathrm{i}} \|} * 100 \qquad with \quad j \in N. \tag{4}$$

Finally, the enrichment process allows us to generate a set of six new features from each weak model, which were used to enrich (extend) the entity dataframe extracted from blockchain data.

In this paper, we implemented the first three classifiers (weak) as Random Forest (RF) and the final one as Gradient Boosting (GB) model, since such implementation has been shown to yield good classification performance in terms of Precision, Recall, and F1-score for entity classification [10]. Specifically, all Random Forest models were implemented with the number of estimators set to 10, with a Gini function to measure the quality of the split and without a maximum depth of the tree. The Gradient Boosting model was implemented with the number of estimators set to 100, the learning rate was set to 0.1, and the maximum depth for limiting the number of nodes was set to 3.

In order to unveil patterns in entity behavior and detect the similarity among training and test batches, we evaluated the classification performance in terms of Precision, Recall, and F1-score calculated per class:

- *Precision* is the number of true positives over the total number of true positives plus false positives. It represents a measure of a classifier's exactness given as a value between 0 and 1, with 1 relating to high precision;
- *Recall* is the number of true positives over the number of true positives plus false negatives. It represents a measure of a classifier's completeness given as a value between 0 and 1;
- F_1-*score* is the harmonic mean of Precision and Recall. It takes values between 0 and 1, with 1 relating to perfect Precision and Recall, and is calculated with Equation (5);

$$F_1 score = 2 * \frac{Precision * Recall}{Precision + Recall}. \tag{5}$$

4.5. K-Fold Cross-Testing

In this study, we aimed to analyze how entity behavior changes over time. The idea is to detect similarity between the distinct temporal batches, represented by good classification performance that indicates the presence of a recurrent pattern. To achieve this goal, we apply a type of "k-fold cross-testing", which allows us to estimate the performance of the implemented cascading machine learning approach. This procedure recalls concepts from k-fold cross-validation and applies them to testing as well, hence the chosen name. In particular, this method allows us to evaluate similarity among different batches generating unique output values for each trained model.

In "k-fold cross-testing", we divided the dataset into k folds (or batches), as explained in the previous section. This batch creation was done without shuffling the data because in Bitcoin analysis it is important to maintain the sequentiality of the data. The number of the generated folds was computed through Equation (2), which generated respectively 3-fold cross-testing for the first experiment, 6-fold cross-testing for the second one, 9-fold cross-testing for the third experiment, and 18-fold cross-testing for the last one. In general, the k-fold cross-testing approach generates a total of k distinct models, each of them obtained by training a system with just one batch (i.e., each fold is used exactly one time for training a model). Then, each model is tested with the remaining $k - 1$ batches left out during the training, as shown in Figure 4. The "k-fold cross-testing" generates k outputs, one for each trained model. A single output represents the average and the standard deviation of the $k - 1$ tests computed over the same trained model.

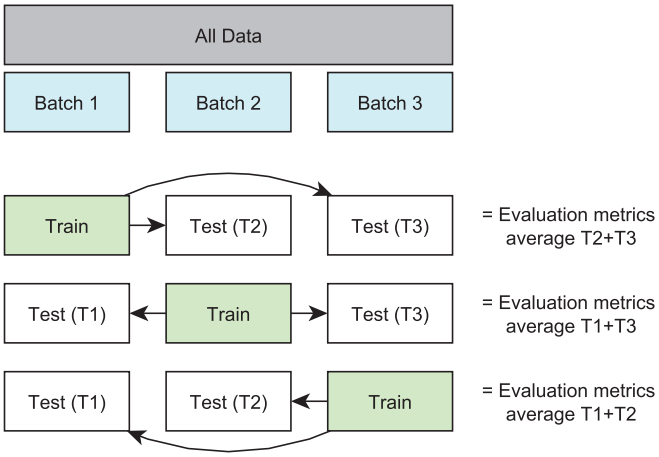


Figure 4. Example of k-fold cross-testing applied with k = 3.

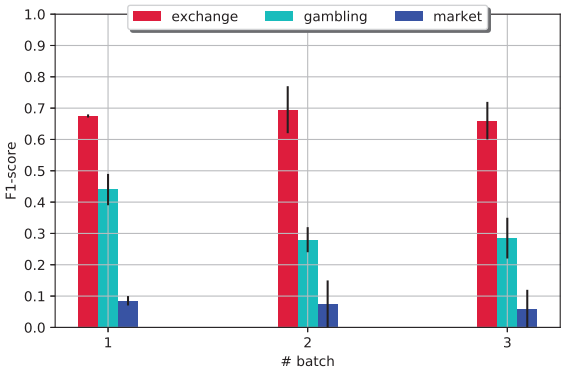
5. Results

In this paper, we present an analysis for evaluating similarity between behavioral patterns of Bitcoin entities using cascading machine learning. The presented process introduces a "temporal view" of the blockchain data aiming to evaluate short-term changes in Bitcoin entity behavior. The approach consists of dividing the whole blockchain into several batches, in order to reduce the complexity of the problem, reducing the amount of data, simplifying the implemented models, and increasing the transferability of the solution.

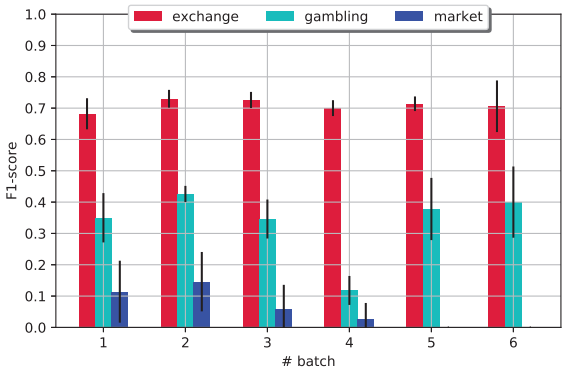
In order to unveil behavioral patterns in the Bitcoin network and study how the size of the batches affects the final classification, four experiments are presented. Each experiment was achieved with a different value of batch size, respectively with 60,000 blocks (approx. 12 months), 30,000 (approx. six months), 20,000 (approx. four months) and 10,000 blocks (approx. two months). The discovered behavioral patterns were divided into six classes: Exchange, Gambling, Market, Mining Pool, Mixer, and eWallet.

Figures 5 and 6 show the F1-score calculated in each experiment separated per class. In particular, in each graph, the x-axis represents the number of batch used to train the models and the y-value

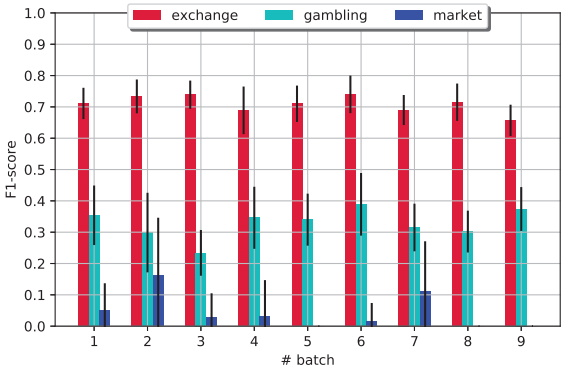
represents the F1-score computed using "k-fold cross-testing" (Section 4.5). The F1-score values are shown with their own standard deviation.



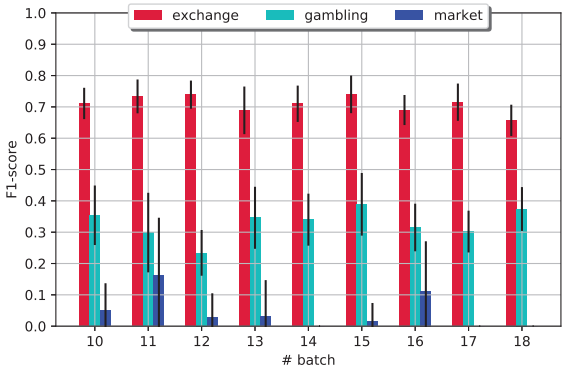
(**a**) Batch size = 60,000



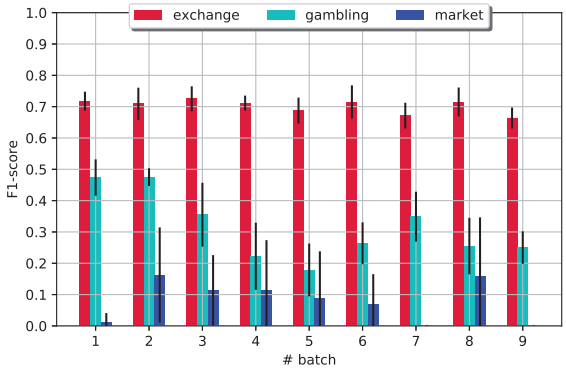
(**b**) Batch size = 30,000



(**c**) Batch size = 10,000 (1st to 9th test batches)

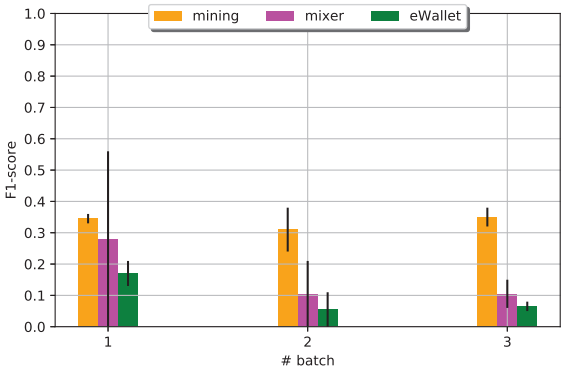


(**d**) Batch size = 10,000 (10th to 18th test batches)

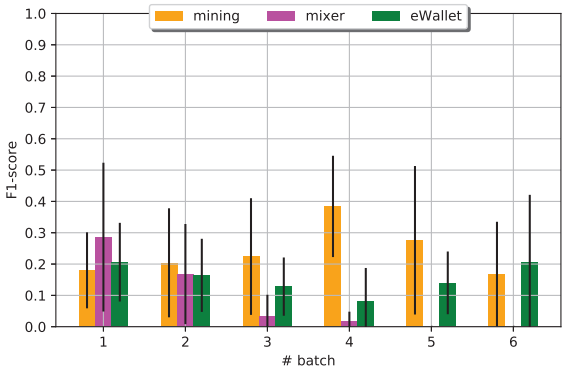


(**e**) Batch size = 20,000

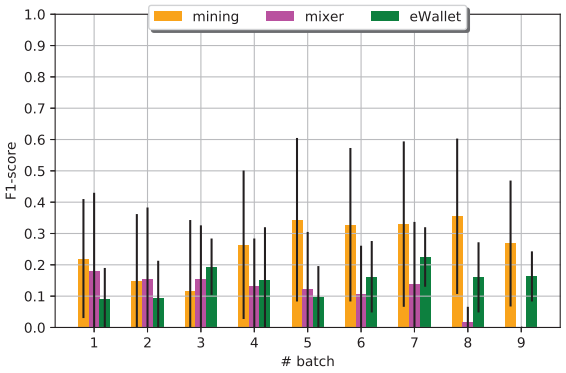
Figure 5. F1-score computed in the four presented experiments with different batch_size for Exchange, Gambling, and Market classes.



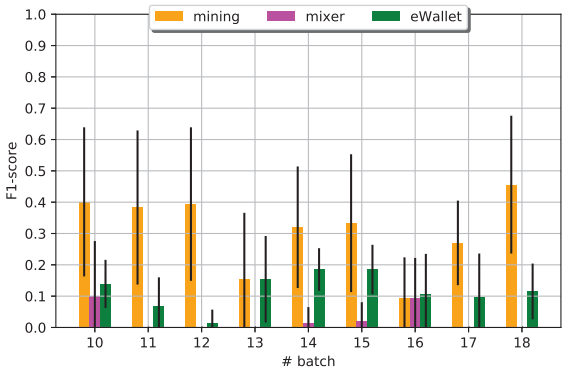
(**a**) Batch size = 60,000



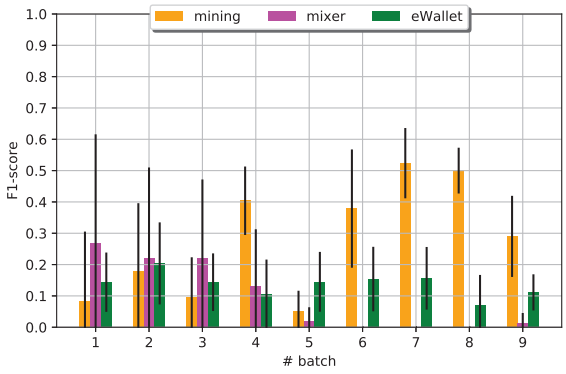
(**b**) Batch size = 30,000



(**c**) Batch size = 10,000 (1st to 9th test batches)



(**d**) Batch size = 10,000 (10th to 18th test batches)



(**e**) Batch size = 20,000

Figure 6. F1-score computed in the four presented experiments with different batch_size for Mining Pool, Mixer, and eWallet classes.

The graphs in Figures 5 and 6 indicate that Exchange, Gambling, and eWallet behavior are only slightly affected by the chosen batch size. In fact, decreasing the batch size, the average F1-score over samples belonging to these three classes is weakly increased. At the same time, smaller batch size penalizes the ability of the models to discover Mixer and Market elements and generates strong changes in the detection of the Mining Pool class. Behavioral changes of those Mining Pools become visible when decreasing the batch size to 20,000 blocks, for example.

These trends are confirmed in Table 3, where the overall average F1-score and standard deviation are reported for each experiment. One can observe that reducing the batch size slightly increases the detection of Exchange and Gambling elements. In particular, considering data from 10,000 blocks

(fourth experiment), F1-score values of 0.72 and 0.39, respectively, were reached. However, the batch size reduction also increases the variability of the classification, generating high standard deviation values for four of the six considered classes. The highest values of standard deviation are obtained for classifying Mining Pool and Market elements. These two classes generate a standard deviation of 0.218 for a F1-score value of 0.29, and 0.112 for a F1-score value of 0.08, respectively.

Table 3. F1-score average and standard deviation average computed with data from each experiment.

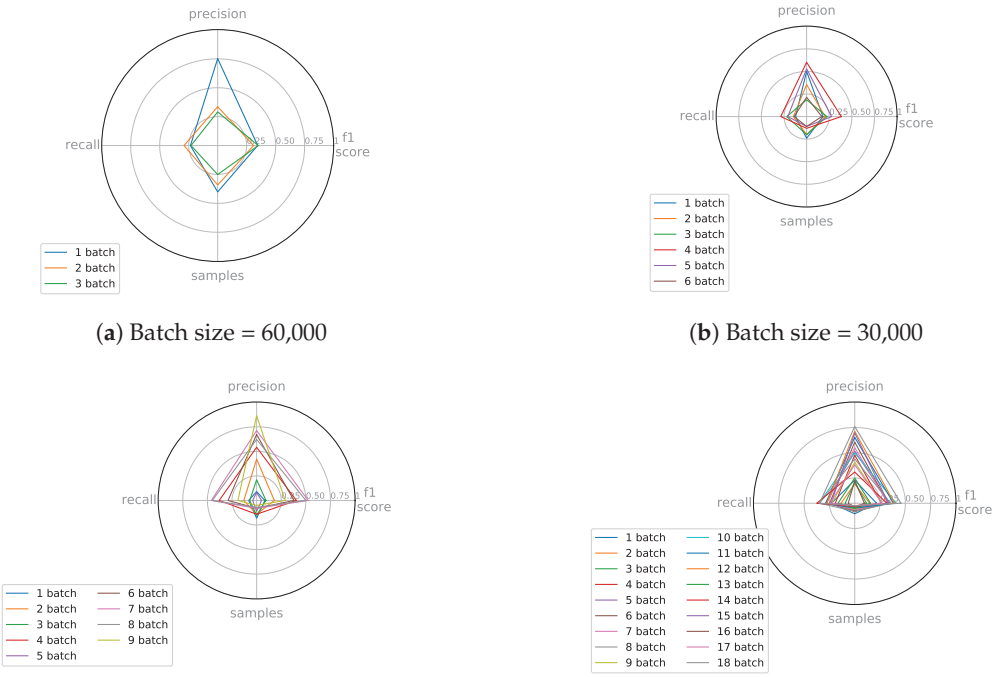
Class	Batch 60,000		Batch 30,000		Batch 20,000		Batch 10,000	
	F1 Average	std. dev. Average	F1 Average	std. dev. Average	F1 Average	std. dev. Average	F1 Average	std. dev. Average
Exchange	0.68	0.047	0.71	0.039	0.70	0.040	0.72	0.050
Gambling	0.34	0.052	0.34	0.071	0.32	0.070	0.39	0.084
Market	0.07	0.050	0.06	0.054	0.08	0.099	0.08	0.112
Mining Pool	0.34	0.038	0.24	0.175	0.28	0.138	0.29	0.218
Mixer	0.16	0.143	0.08	0.083	0.10	0.128	0.07	0.099
eWallet	0.10	0.037	0.15	0.126	0.14	0.098	0.13	0.102

The results shown in Figure 6 and Table 3 motivated us to analyze more in-depth the strong changes that could be observed in the Mining Pool class detection, which, following our initial idea, relate to changes in behavioral patterns that only become visible when reducing the batch size. For batch size 20,000, for example, Mining Pool behavior trained with temporal batch 7 yielded on average much better classification results than the model trained on other batches. This shows that the Mining Pool behavior captured in batch 7 is more representative than the behavior defined in all other batches.

Figure 7 shows detailed associations among F1-score, recall, precision, and normalized number of samples for the Mining Pool behavior for each batch and batch size as Radar graphs. In particular, the highest values of recall and F1-score were obtained with batch size set to 20,000 blocks training the models with batch number 7 (0.46 and 0.52, respectively), while the highest values of precision is obtained by training the models with batch number 9 (0.86), as shown in Figure 7c. Decreasing the batch size down to 10,000 generates a linear variance in precision between 0.25 and 0.75, keeping the F1-score in range 0.25–0.50 (Figure 7d).

Figure 8 provides another interesting picture of the Mining Pool behavior. These graphs represent confusion matrices where the values of shown F1-scores were not averaged but represent the actual values computed for each test run. Each cell-value represents the F1-score of the models trained with the i-th batch and tested with the j-th batch. In case of $i = j$, the value was not computed setting the cell to -1 (due to this, they are shown in black).

Using heatmaps [48], it becomes clear that decreasing the batch size generates an improvement in terms of F1-score in the detection of Mining Pool behavioral patterns (Figure 8a–d). In the first experiment, there were 0 elements with F1-score above 0.70 (a threshold that we chose to indicate a good classification). In fact, the highest value was 0.38. In the second experiment, there was just one test with F1-score above the chosen threshold reaching the value of 0.71. In the third experiment, no values above the threshold were achieved; however, the highest value of 0.67 was reached in three different cases. In the fourth experiment (smallest batch size of 10,000), 14 tests presented values above the threshold reaching a maximum value of 0.88. Interestingly, Figure 8c (batch size 20,000) clearly shows how "early" models (trained on temporally earlier blocks) up to batch number 6 do not yield good classification performance in neither previous nor following batches. Models trained on batches 6–8, though, achieved good performance in the first batches highlighting that generally it was more difficult to "predict the future" (earlier models perform poorly for future batches). Moreover, Mining Pool behavior in batch 7, for example, seems to represent a "typical" behavioral pattern that is recurrent over time across several batches.



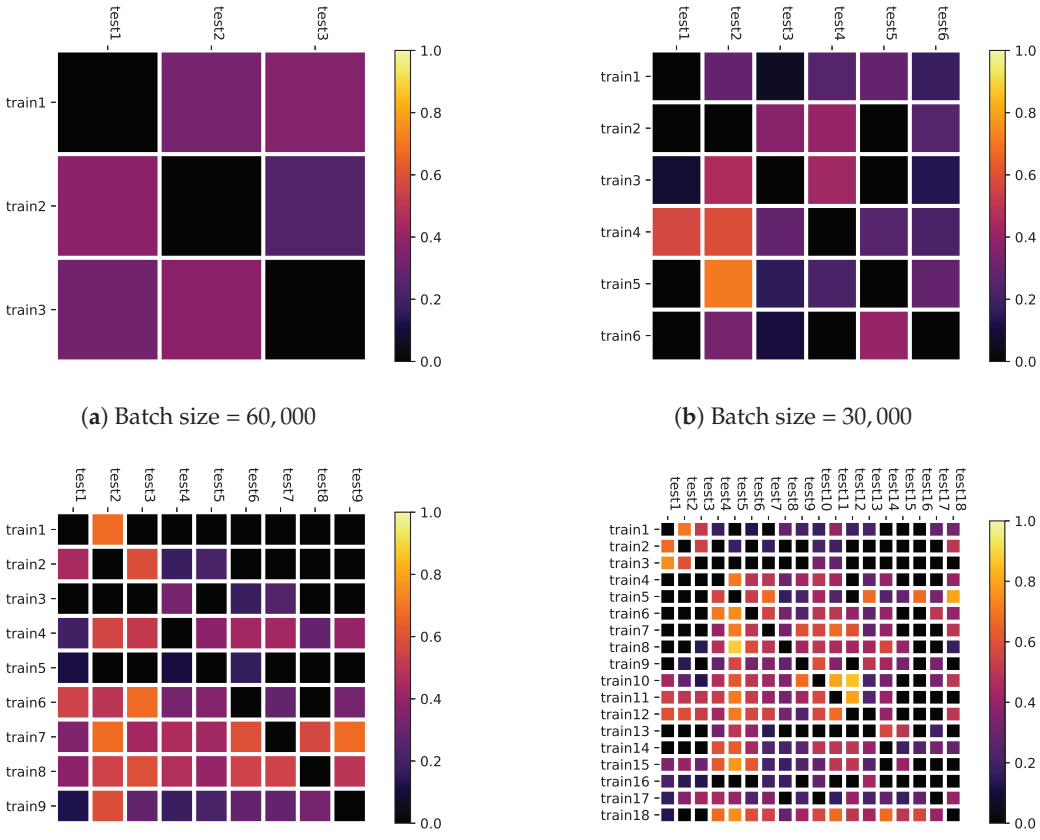
(**a**) Batch size = 60,000

(**b**) Batch size = 30,000

(**c**) Batch size = 20,000

(**d**) Batch size = 10,000

Figure 7. Radar graph with precision, recall, F1-score, and number of samples (normalized) for the Mining Pool class.



(**a**) Batch size = 60, 000

(**b**) Batch size = 30, 000

(**c**) Batch size = 20, 000

(**d**) Batch size = 10, 000

Figure 8. Mining Pool F1-score of each test in the four presented experiments.

6. Discussion

The general aim of this paper was to address problems that are associated with using the entire blockchain for entity classification in machine learning forensics—which are higher complexity (requiring larger computational efforts) and the inherent problem of hiding network micro-dynamics potentially important for detecting short-term changes in entity behavior. We therefore considered Bitcoin blockchain data as time-series and divided it into temporal batches of varying size.

Our study showed that considering the blockchain as a time-series (i.e., dividing Bitcoin data into temporal batches) yields interesting insights into behavioral changes of network entities over time, which may be a starting point for future studies focusing on the investigation of network micro-dynamics.

The key idea behind our approach was that a model trained on a batch of a certain time-interval should achieve good classification performance when testing it on another temporal batch, in case entity behavioral patterns (described by the chosen, extracted features) within those batches is similar. Thus, each temporal batch was used to define and extract entity behavior, which allowed us to analyze how the chosen interval and batch size affects classification performance. Classification methods were based on cascading machine learning principles and "k-fold cross-testing", which implies training models on one batch and testing it on all other batches until all batches have been used for training.

From Section 5, and in particular from Figures 5 and 6, we observed that the choice of batch size did affect all considered classes. Exchange, Gambling, and eWallet classifications were only slightly affected and yielded a weakly increasing trend in F1-score with decreasing batch size. In particular, reducing the batch size (thus considering micro-dynamics within the Bitcoin network) did not show great changes in the final classification of these classes. Moreover, for the Exchange class, this demonstrated that a smaller temporal batch size indeed seemed to be sufficient for Exchange classification—results suggest that it is not necessary to use the full blockchain data to obtain good classification rates.

Market and Mixer classification performance instead worsened when decreasing the batch size. This is reflected in Figure 5e and Figure 6e where F1-scores of these classes reached values close to or equal to 0. In this case, results probably depended on the small sample size of Market and Mixer entities prevalent in smaller batches.

However, the most interesting effect of batch size reduction occurred for the Mining Pool classification. In fact, Mining Pool classification performance changed dramatically when moving to smaller batch sizes, interestingly showing increasing standard deviations of averaged F1-scores for smaller batches, hinting at higher performance differences from batch to batch. These results led us to analyze Mining Pool behavior more in detail—especially as this class represents a sensitive target for cyber-attacks, as presented in [4,13].

Confusion matrices shown in Figure 8a indicate that, for Mining Pool behavior, the best classification was obtained by training models with "recent" data in order to predict "past" behavior when moving to smaller batch sizes, as the majority of the highest F1-scores were below the main diagonal. This suggests that, generally, it was more difficult to "predict the future". In fact, earlier models did not capture future behavior, but recent models did seem to contain behavioral elements that can be seen in the past as well as new elements belonging to more recent data. This could imply a certain development of network behavior over time, using old patterns of which certain features are kept but where new features evolve. Future work could investigate these findings more in detail in other networks as well.

Our findings further highlight that decreasing the batch size (to 10,000, i.e., two months of data, for example) provided a higher "temporal resolution" allowing us to detect behavioral changes that were not visible with larger batch sizes. In particular, results shown in Figure 8d corresponding to the model trained with batch number 12, are interesting. It is possible to observe that the model achieved good classification scores for test batch numbers 1, 2, 3 and 5, but decreased its accuracy in batch 4. This situation could be a symptom of a suspicious activity (or cyber-attacks) of entities in the fourth test batch. In fact, the low score in batch 4 suggests a sudden change of behavior. These kinds of

changes can only be appreciated focusing on a small time interval, as using the whole blockchain data most likely would hide these micro-effects. We believe that such findings for particular entities could be investigated more in detail in future work by cross-referencing them a posteriori with known attacks or malicious activities.

Our results, compared with results presented in [10] (where the whole blockchain is considered), showed lower classification rates across all considered classes (from 30% to 90%). However, as mentioned in Section 1, improving classification rates was not the aim of this study. Our idea was to introduce temporal batch division of blockchain data and aimed to analyze whether this approach may uncover micro-dynamics that determine changes in entity behavior. In fact, sudden, short-term behavioral changes of a fixed entity represented by changes of classification scores could be a symptom of suspicious behavior or an attack. Rather than looking to absolute classification scores, we therefore focused on changes or variability of scores from batch to batch.

Furthermore, it became apparent that batch size affects the detection of changes in entity behavior in different ways. A batch size that is too small seems to create difficulty in micro-dynamics detection in a small dataset (for example for Mixer, Market and Mining Pool), but entity behavior in general (Exchange, Gambling and eWallet) is enhanced.

To the best of our knowledge, this is the first study using temporal batch division of Bitcoin blockchain data, and hence we are unable to perform a comparison against similar approaches. To date, no other study has focused on detecting similarities between entity behavioral patterns in various time batches of Bitcoin blockchain data.

We acknowledge that our approach does not fully get rid of the "black-box model", but we believe that our temporal dissection into "smaller black-boxes" has shown interesting changes in entity pattern behavior and consider it a first step towards investigating Bitcoin behavioral pattern evolution in much more detail.

One major drawback of our approach is the dependence on WalletExplorer data. This dataset allows us to only include well-known entities that were already detected, while special entities or individual users could not be included.

Furthermore, our work provides guidelines which could be used to model a "typical" Mining Pool and, in particular, we plan to use this information to develop a model to detect abnormal or illicit behavior and behavioral changes posing possible security threats—for example, by exploiting results presented in [49], where it is stated that a type of cyber-attack produces short-term effects in the Mining Pool itself.

Future development could also be aimed at providing customization of behavioral patterns and simulating them in a controlled environment. This would allow researchers to replicate "typical" entity behavior or specific types of attacks or illicit activities related to certain entities. Furthermore, the controlled environment, for example the one presented in [50], could be used to detect limitations of the protocol, creating specific scenarios or defining new entity behavior not yet discovered in the mainnet.

7. Conclusions

We conclude that the right choice of time interval size helped reduce the complexity for some classes and helped with the detection of abnormal/interesting activities and behavioral changes for other classes of Bitcoin entities. Due to these properties, we believe that the presented analysis can be a starting point to significantly improve some forensics tools that currently apply machine learning on the whole blockchain—providing novel, more in-depth insight into what is happening within the Bitcoin network by carefully choosing the "temporal resolution" of the analysis.

Author Contributions: Conceptualization, F.Z., J.L.B., and R.O.U.; Related Work, F.Z.; Datasets, F.Z. and M.E.; Methodology, F.Z.; Results, F.Z.; Discussion, F.Z., J.L.B., and M.G.; Writing—original draft preparation, F.Z.; Writing—review and editing, J.L.B. and M.G.; Supervision, R.O.U.

Funding: This work was partially funded by the European Commission through the Horizon 2020 research and innovation program, as part of the "TITANIUM" project (Grant Agreement No. 740558).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

BTC	Bitcoin
CEO	Chief Executive Officer
COO	Chief Operating Officer
DDoS	Distributed Denial-of-Service
DoS	Denial–of–Service
GB	Gradient Boosting
HMM	Hidden Markov Model
LEO	Law Enforcement Officer
RF	Random Forest
STPN	Spatio-Temporal Pattern Network
USD	United States Dollar

References

1. Nakamoto, S. *Bitcoin: A Peer-to-Peer Electronic Cash System*; Bitcoin: Saint Christopher, saint Christopher and nevis, 2008.
2. Crosby, M.; Pattanayak, P.; Verma, S.; Kalyanaraman, V. Blockchain technology: Beyond bitcoin. *Appl. Innov.* **2016**, *2*, 71.
3. Narayanan, A.; Bonneau, J.; Felten, E.; Miller, A.; Goldfeder, S. *Bitcoin and Cryptocurrency Technologies*; Princeton University Press: New Jersey, NJ, USA, 2016.
4. Vasek, M.; Thornton, M.; Moore, T. Empirical analysis of denial-of-service attacks in the Bitcoin ecosystem. In Proceedings of the International Conference on Financial Cryptography and Data Security, Christ Church, Barbados, 3–7 March 2014; pp. 57–71.
5. Bitcoin Accounts for 95% of Cryptocurrency Crime, Says Analyst. Available online: https://fortune.com/2019/04/24/bitcoin-cryptocurrency-crime/ (accessed on 19 November 2019).
6. Building Trust in Blockchains. Available online: https://www.chainalysis.com (accessed on 19 November 2019).
7. Kethineni, S.; Cao, Y. The Rise in Popularity of Cryptocurrency and Associated Criminal Activity. *Int. Crim. Justice Rev.* **2019**. [CrossRef]
8. Fanusie, Y.; Robinson, T. *Bitcoin Laundering: An Analysis of Illicit Flows Into Digital Currency Services*; Elliptic: London, UK, 2018.
9. Conti, M.; Kumar, E.S.; Lal, C.; Ruj, S. A survey on security and privacy issues of bitcoin. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 3416–3452. [CrossRef]
10. Zola, F.; Eguimendia, M.; Bruse, J.L.; Orduna Urrutia, R. Cascading Machine Learning to Attack Bitcoin Anonymity. In Proceedings of the 2nd IEEE International Conference on Blockchain, Atlanta, GA, USA, 14–17 July 2019; pp. 1–8.
11. Jourdan, M.; Blandin, S.; Wynter, L.; Deshpande, P. Characterizing Entities in the Bitcoin Blockchain. *arXiv* **2018**, arXiv:1810.11956.
12. Harlev, M.A.; Sun Yin, H.; Langenheldt, K.C.; Mukkamala, R.; Vatrapu, R. 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, Waikoloa Village, HI, USA, 3–6 January 2018.
13. Johnson, B.; Laszka, A.; Grossklags, J.; Vasek, M.; Moore, T. Game-theoretic analysis of DDoS attacks against Bitcoin mining pools. In Proceedings of the International Conference on Financial Cryptography and Data Security, Christ Church, Barbados, 3–7 March 2014; pp. 72–86.
14. Abhishta, A.; Joosten, R.; Dragomiretskiy, S.; Nieuwenhuis, L.J. Impact of Successful DDoS Attacks on a Major Crypto-currency Exchange. In Proceedings of the 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Pavia, Italy, 13–15 February 2019; pp. 379–384.

15. Paquet-Clouston, M.; Haslhofer, B.; Dupont, B. Ransomware payments in the bitcoin ecosystem. *J. Cybersecur.* **2019**, *5*, 1–11. [CrossRef]
16. Binance Security Breach Update. Available online: https://binance.zendesk.com/hc/en-us/articles/360028031711-Binance-Security-Breach-Update (accessed on 19 November 2019).
17. Top 100 Cryptocurrencies by Market Capitalization. Available online: https://coinmarketcap.com (accessed on 19 November 2019).
18. Böhme, R.; Christin, N.; Edelman, B.; Moore, T. Bitcoin: Economics, technology, and governance. *J. Econ. Perspect.* **2015**, *29*, 213–38. [CrossRef]
19. Meiklejohn, S.; Orlandi, C. Privacy-enhancing overlays in bitcoin. In Proceedings of the International Conference on Financial Cryptography and Data Security, San Juan, Puerto Rico, 26–30 January 2015; pp. 127–141.
20. Androulaki, E.; Karame, G.O.; Roeschlin, M.; Scherer, T.; Capkun, S. Evaluating user privacy in bitcoin. In Proceedings of the International Conference on Financial Cryptography and Data Security, Okinawa, Japan, 1–5 April 2013; pp. 34–51.
21. Herrera-Joancomarti, J. Research and challenges on bitcoin anonymity. In *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*; Springer: Wroclaw, Poland, 2015; pp. 3–16.
22. Liao, K.; Zhao, Z.; Doupé, A.; Ahn, G.J. Behind closed doors: Measurement and analysis of CryptoLocker ransoms in Bitcoin. In Proceedings of the 2016 APWG Symposium on Electronic Crime Research (eCrime), Toronto, ON, Canada, 1–3 June 2016; pp. 1–13.
23. Koshy, P.; Koshy, D.; McDaniel, P. An analysis of anonymity in bitcoin using p2p network traffic. In Proceedings of the International Conference on Financial Cryptography and Data Security, Christ Church, Barbados, 3–7 March 2014; pp. 469–485.
24. Reid, F.; Harrigan, M. An analysis of anonymity in the bitcoin system. In *Security and Privacy in Social Networks*; Springer: New York, NY, USA, 2013; pp. 197–223.
25. Fleder, M.; Kester, M.S.; Pillai, S. Bitcoin transaction graph analysis. *arXiv* **2015**, arXiv:1502.01657.
26. Pham, T.; Lee, S. Anomaly detection in bitcoin network using unsupervised learning methods. *arXiv* **2016**, arXiv:1611.03941.
27. Monamo, P.; Marivate, V.; Twala, B. Unsupervised learning for robust Bitcoin fraud detection. In Proceedings of the 2016 Information Security for South Africa (ISSA), Johannesburg, South Africa, 17–18 August 2016; pp. 129–134.
28. Hirshman, J.; Huang, Y.; Macke, S. *Unsupervised Approaches to Detecting Anomalous Behavior in The Bitcoin Transaction Network*, 3rd ed.; Technical Report; Stanford University: Stanford, CA, USA, 2013.
29. Bartoletti, M.; Pes, B.; Serusi, S. Data mining for detecting Bitcoin Ponzi schemes. In Proceedings of the 2018 Crypto Valley Conference on Blockchain Technology (CVCBT), Zug, Switzerland, 20–22 June 2018; pp. 75–84.
30. Yin, H.S.; Vatrapu, R. A first estimation of the proportion of cybercriminal entities in the bitcoin ecosystem using supervised machine learning. In Proceedings of the 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 11–14 December 2017; pp. 3690–3699.
31. Wolpert, D.H. Stacked generalization. *Neural Netw.* **1992**, *5*, 241–259. [CrossRef]
32. Ranshous, S.; Joslyn, C.A.; Kreyling, S.; Nowak, K.; Samatova, N.F.; West, C.L.; Winters, S. Exchange pattern mining in the bitcoin transaction directed hypergraph. In Proceedings of the International Conference on Financial Cryptography and Data Security, Sliema, Malta, 3–7 April 2017; pp. 248–263.
33. Basseville, M.; Nikiforov, I.V. *Detection of Abrupt Changes: Theory and Application*; Prentice Hall Englewood Cliffs: Upper Saddle River, NJ, USA, 1993; Volume 104.
34. Hoang, X.; Hu, J. An efficient hidden Markov model training scheme for anomaly intrusion detection of server applications based on system calls. In Proceedings of the 2004 12th IEEE International Conference on Networks (ICON 2004), Singapore, 16–19 November 2004; Volume 2, pp. 470–474.
35. Aggarwal, C.C.; Zhao, Y.; Philip, S.Y. Outlier detection in graph streams. In Proceedings of the 2011 IEEE 27th International Conference on Data Engineering, Hannover, Germany, 11–16 April 2011; pp. 399–409.
36. Mitrović, S.; Baesens, B.; Lemahieu, W.; De Weerdt, J. Churn prediction using dynamic rfm-augmented node2vec. In Proceedings of the International Workshop on Personal Analytics and Privacy, Dublin, Ireland, 10–14 September 2017; pp. 122–138.

37. Eberle, W.; Holder, L. Incremental Anomaly Detection in Graphs. In Proceedings of the 2013 IEEE 13th International Conference on Data Mining Workshops, Dallas, TX, USA, 7–10 December 2013; pp. 521–528.
38. Blazek, R.B.; Kim, H.; Rozovskii, B.; Tartakovsky, A. A novel approach to detection of denial-of-service attacks via adaptive sequential and batch-sequential change-point detection methods. In Proceedings of IEEE Systems, Man and Cybernetics Information Assurance Workshop, West Point, NY, USA, 5–6 June 2001
39. Bitcoin Block Explorer with Address Grouping and Wallet Labeling. Available online: https://www.walletexplorer.com/ (accessed on 19 November 2019).
40. Mainnet, Bitcoin Main Network. Available online: https://bitcoin.org/en/glossary/mainnet (accessed on 19 November 2019).
41. Samsudeen, Z.; Perera, D.; Fernando, M. Behavioral Analysis of Bitcoin Users on Illegal Transactions. *Adv. Sci. Technol. Eng. Syst. J.* **2019**, *4*, 402–412. [CrossRef]
42. Download Bitcoin Core. Available online: https://bitcoin.org/en/download (accessed on 19 November 2019).
43. Protect Your Privacy. Available online: https://bitcoin.org/en/protect-your-privacy (accessed on 19 November 2019).
44. Ermilov, D.; Panov, M.; Yanovich, Y. Automatic Bitcoin address clustering. In Proceedings of the 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), Cancun, Mexico, 18–21 December 2017; pp. 461–466.
45. Lacroix, V.; Fernandes, C.G.; Sagot, M.F. Motif search in graphs: Application to metabolic networks. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **2006**, *3*, 360–368. [CrossRef] [PubMed]
46. Van der Laan, M.J.; Polle, E.C.; Hubbard, A.E. Super learner. *Stat. Appl. Genet. Mol. Biol.* **2007**, *6*. [CrossRef] [PubMed]
47. Zhou, Z.H. *Ensemble methods: Foundations and Algorithms*; Chapman and Hall: London, UK; CRC: Boca Raton, FL, USA, 2012.
48. seaborn.heatmap. Available online: https://seaborn.pydata.org/generated/seaborn.heatmap.html (accessed on 19 November 2019).
49. Laszka, A.; Johnson, B.; Grossklags, J. When bitcoin mining pools run dry. In Proceedings of the International Conference on Financial Cryptography and Data Security, San Juan, Puerto Rico, 26–30 January 2015; pp. 63–77.
50. Zola, F.; Pérez-Solà, C.; Zubia, J.E.; Eguimendia, M.; Herrera-Joancomartí, J. Kriptosare. gen, a Dockerized Bitcoin Testbed: Analysis of Server Performance. In Proceedings of the 2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Canary Islands, Spain, 24–26 June 2019; pp. 1–5.

2. *Network traffic analysis through node behaviour classification: a graph-based approach with temporal dissection and data-level preprocessing*

The work related to this part is:

- Zola F., Segurola Gil L., Bruse J.L., Galar M. and Orduna Urrutia R., *Network traffic analysis through node behaviour classification: a graph-based approach with temporal dissection and data-level preprocessing*, Computers & Security, 115 (2022): 102632, DOI: 10.1016/j.cose.2022.102632

- Status: Published.
- Impact Factor (JCR 2020): 4.438.
- Knowledge area:
  - Computer Science, Information Systems. Ranking 40/161 (Q1).

Computers & Security 115 (2022) 102632

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Network traffic analysis through node behaviour classification: a graph-based approach with temporal dissection and data-level preprocessing☆

F. Zola[a,b,1,*], L. Segurola-Gil[a,2], J.L. Bruse[a,3], M. Galar[b,4], R. Orduna-Urrutia[a,5]

[a] Vicomtech Foundation, Basque Research and Technology Alliance (BRTA) Paseo Mikeletegi 57, Donostia/San Sebastian 20009, Spain
[b] Institute of Smart Cities, Public University of Navarre, Pamplona 31006, Spain

ARTICLE INFO

Article history:
Received 28 July 2021
Revised 4 January 2022
Accepted 27 January 2022
Available online 29 January 2022

Keywords:
Network analysis
Graph imbalance problem
Graph Convolutional Networks
Temporal dissection
Deep Learning

ABSTRACT

Network traffic analysis is an important cybersecurity task, which helps to classify anomalous, potentially dangerous connections. In many cases, it is critical not only to detect individual malicious connections, but to detect which *node* in a network has generated malicious traffic so that appropriate actions can be taken to reduce the threat and increase the system's cybersecurity. Instead of analysing connections only, node behavioural analysis can be performed by exploiting the graph information encoded in a connection network. Network traffic, however, is temporal data and extracting graph information without a fixed time scope may only unveil macro-dynamics that are less related to cybersecurity threats. To address these issues, a threefold approach is proposed here: firstly, temporal dissection for extracting graph-based information is applied. As the resulting graphs are typically affected by class imbalance (i.e. malicious nodes are under-represented), two novel graph data-level preprocessing techniques - *R-hybrid* and *SM-hybrid* - are introduced, which focus on exploiting the most relevant graph substructures. Finally, a Neural Network (NN) and two Graph Convolutional Network (GCN) approaches are compared when performing node behaviour classification. Furthermore, we compare the node classification performance of these supervised models with traditional unsupervised anomaly detection techniques. Results show that temporal dissection parameters affected classification performance, while the data-level preprocessing strategies reduced class imbalance and led to improved supervised node behaviour classification, outperforming anomaly detection models. In particular, Neural Network (NN) outperformed Graph Convolutional Network (GCN) approaches for two attack families and was less affected by class imbalance, yet one GCN performed best overall. The presented study successfully applies a temporal graph-based approach for malicious actor detection in network traffic data.

1. Introduction

Today, cyber attacks are becoming more sophisticated and more destructive than ever. Attackers increasingly try to find vulnerabilities and exploit breaches in order to gain unauthorized access, damage or steal information, assets, network or any kind of sensitive data. If a cyber attack compromises at least one of the three security pillars of a target - confidentiality, integrity or availability - it can generate a considerable loss of value for the owner, in economical, ethical, digital, psychological and societal terms (Agrafiotis et al., 2018; Formosa et al., 2021).

Currently, increasing efforts are being made to prevent such cyber threats and reduce their impact (Khraisat et al., 2019; Van Schaik et al., 2020). However, as it is difficult to determine rules for manual or semi-automatic threat detection, there is a

☆ This work has been partially supported by the Spanish Centre for the Development of Industrial Technology (CDTI) under the project ÉGIDA (EXP 00122721 / CER-20191012) - RED DE EXCELENCIA EN TECNOLOGIAS DE SEGURIDAD Y PRIVACIDAD.

* Corresponding author.
E-mail addresses: fzola@vicomtech.org (F. Zola), lsegurola@vicomtech.org (L. Segurola-Gil), jbruse@vicomtech.org (J.L. Bruse), mikel.galar@unavarra.es (M. Galar), rorduna@vicomtech.org (R. Orduna-Urrutia).
[1] orcid:0000-0002-1733-5515.
[2] orcid:0000-0003-4278-9081.
[3] orcid:0000-0002-5774-1593.
[4] orcid:0000-0003-2865-6549.
[5] orcid:0000-0002-5932-0987.

https://doi.org/10.1016/j.cose.2022.102632

need for new solutions based on Artificial Intelligence (Xin et al., 2018). Derived techniques such as Deep Learning have shown promising results in terms of quantifying cyber risks and optimizing cybersecurity operations (Sarker et al., 2020).

Machine learning (ML) techniques are often used to perform anomaly detection (AD) (Omar et al., 2013), an operation that can be used for both threat detection and threat prevention. In particular, anomaly detection tasks considering time series analyses have been extensively explored in recent years (Li et al., 2021). However, in a complex system of actors or entities, interaction patterns can evolve as entities may disappear, emerge or simply change their dynamics. Considering network traffic information in a graph-based representation can highlight these changes and favors the definition, classification and visualization of such information (Akoglu et al., 2015; Tan et al., 2020) as graphs integrate both structured and unstructured data in a representation of entities and their relationships (Zhou et al., 2009). However, approaches that apply Deep Learning paradigms directly to graphs have so far mainly been applied in other domains (Kipf and Welling, 2017; Scarselli et al., 2008).

The majority of graph-based studies in the cybersecurity domain are focused on identifying anomalous or malicious traffic by directly classifying node communications using statistical (Djidjev et al., 2011), data mining (Iliofotou et al., 2009b) or machine learning (Yao et al., 2018; Zheng et al., 2019) techniques. However, analyzing every single connection in real networks may be resource-consuming due to the large amount of flows to be processed. Furthermore, malformed communications carried out by normal entities can result in falsely detected malicious flows (false positives), and attackers could generate normal activities in a complex attack routine for obfuscating their presence in the network.

In these cases, it is interesting to classify a single connection and at the same time detect who has generated it, in order to make appropriate decisions that may increase network cybersecurity (i.e. exclude, isolate entities etc.). For these reasons, we introduce here a novel methodology that shifts the focus from analyzing the edges of a graph (nodes communication) to its nodes (entities behaviour) for malicious/attack detection. More concretely, this novel methodology intends to translate an attack classification task into a (node) behavioural classification task, thus converting time series network traffic into graph-based structures. Our approach is presented in three distinct phases: *graph creation, data-level preprocessing,* and finally, *classification*.

In the first phase, the main idea is to split the entire network information into temporal intervals and extract graph-based structures from each of them, recalling the concept of a temporal Traffic Dispersion Graph (TDG) (Iliofotou et al., 2009a). In this way, it is possible to visualize entity interactions and their dynamics over time. Furthermore, this temporal dissection avoids small, scarce interactions between entities being obfuscated by more frequent ones. In fact, considering the whole graph at once can generate skewed scenarios that may encourage machine learning models to classify macro-dynamics over more critical micro-dynamics. Further, long times would be required to detect attack activities, making macro-dynamics approach less usable.

After extracting the graphs - as in almost any classification problem related to cybersecurity - it is important to deal with class imbalance (Japkowicz and Stephen, 2002) since typically normal network activities outnumber anomalous activities or attacks. This phenomena can strongly degrade the quality of classification performance, especially when supervised machine learning techniques are used (Fernández et al., 2018). Since common data-level methods for class imbalance cannot directly be applied to graph data, new approaches are required. For this reason, in the second phase, we propose two novel approaches for dealing with class imbalance in graph structures, which we call *R-hybrid* and *SM-hybrid*. The key idea is to apply operations of data-level preprocessing to graph structures, while avoiding the distortion of their topology.

In phase three, behavioural classification tasks using the newly balanced graph-based structures are performed. The main goal of this phase is to compare the performance of models that are able to exploit both behavioural features and graph relations (such as Graph Convolutional Networks, GCNs), with the ones using behavioural features only (such as Neural Networks, NNs). In this way, it is possible to analyze how much the information related to graph connections affects the classification task.

Furthermore, in order to explore the benefits and limitations of our approach, we compare classification results achieved by training graph-based supervised models on the balanced dataset with several common state-of-the-art unsupervised anomaly detection techniques. These unsupervised models are trained with the imbalanced dataset as their main goal is to detect anomalies in the data distribution (outliers) (Goldstein and Uchida, 2016), (Leung and Leckie, 2005). A relevant cybersecurity dataset (Moustafa and Slay, 2015) containing network traffic data is used to validate the presented approach.

To the best of our knowledge, this is the first work that proposes an entire pipeline for working with network traffic data extracting temporal graph-based (node) behaviours, handling class imbalance in graph structures and finally performing classification to detect malicious entities/nodes.

The rest of the paper is organized as follows. Section 2 introduces several key concepts concerning graph theory and then presents related work in terms of graph construction techniques, class imbalance and Deep Learning on graphs. In Section 3, motivations and contributions are described as well as our novel approach and its phases. Section 4 provides an overview of the used datasets and their limitations, the metrics used to evaluate the experiments, the extracted graphs and presents the experiments. Section 5 describes and discusses the obtained classification results. Finally, conclusions and guidelines for future work are given in Section 6.

2. Preliminaries

In this section, theoretic concepts applied in this study are introduced. In particular in Section 2.1, key concepts related to graph theory are presented, whereas in Section 2.2 related works are described.

2.1. Graph Theory

The aim of this study is to classify node behaviours extracted from network traffic and analyze them in a graph domain. For this reason, key concepts related to graph theory are reported, following their definitions in Bollobás (2013).

Definition 2.1. A *graph* G is defined as the ordered pair $G = (V, E)$, where V represents vertex or node set and E is an unordered pair of elements of V called the set of edges. The number of nodes and edges for G will be $|V|$ and $|E|$ respectively.

Definition 2.2. Let $G = (V, E)$ be a graph. Two vertices $u, v \in V$ are *incident* with the edge $e_i \in E$ iff $e_i = \{u, v\}$. A *walk* consists of an alternating sequence of consecutive incident vertices and edges that begins and ends with a vertex. A *path* is a walk without repeated vertices.

Definition 2.3. Let $G = (V, E)$ be a graph. Then, $G' = (V', E')$ is a *subgraph* of G iff $V' \subseteq V$ and $E' \subseteq E$. It can be written as $G' \subseteq G$.

Definition 2.4. Let $G = (V, E)$ be a graph. G is *connected* iff $\forall u, v \in V$, $\exists$ a sequence of edges $e_1, e_2, ..., e_n \in E$ such that there is a path

from u to v. If a graph is not connected, every connected maximal subgraph of it is called a *component* of the graph.

Definition 2.5. Let $G = (V, E)$ be a graph and let $e \in E$ be an edge of the graph. Let $G' = (V, E - \{e\})$ be the subgraph of G with e cut. Let $c(G)$ denote the number of components of the graph G. Then, e is said to be a *bridge* of G iff $c(G') = c(G) + 1$.

In this work, the focus is on graphs and its components. Therefore, and abusing the notation, the term *subgraph* is used for referring to each component of a graph for the rest of this paper.

2.2. Related Work

In this section, related work is introduced, organized as follows: in Section 2.2.1 techniques for extracting graph information from time series are analyzed, while in Section 2.2.2, techniques for addressing class imbalance are presented. In Section 2.2.3, related work based on Graph Convolutional Networks is described, and finally in Section 2.2.4, models used for anomaly detection are introduced

2.2.1. Graph Construction Techniques from Time Series

Network traffic datasets are composed of information gathered from a network that is usually represented as time series. Each row in these datasets contains information and features related to packages, connections or flows between a source and a destination (depending on the granularity of the dataset). For simplicity, we use the word *flow* to indicate a row of any network traffic dataset.

The first step is to convert this time series information into temporal graph structures. However, domain transformation can always cause distortion or loss of information. These problems can be even more severe when converting unstructured data or a time series into a network data representation. In Silva and Zhao (2016), several techniques for transforming unstructured data, vector-based data, or even time series data into networks are described.

In Wehmuth et al. (2015), a model for representing finite discrete Time-Varying Graphs (TVGs, typically used to model complex dynamic network systems), is presented, whereas in Crovella and Kolaczyk (2003) a graph wavelet approach is applied on elements connected via an arbitrary graph topology (spatial domain). In Djidjev et al. (2011) and Jin et al. (2009), temporal graph representations for analyzing large networks are introduced, using telescoping subgraphs (TSGs) in the first case and Traffic Activity Graphs (TAGs), in the second case.

An alternative was presented by Ilioftou et al Iliofotou et al. (2007), known as Traffic Dispersion Graph (TDG). These graphs are graphical representations of various interactions of a group of nodes ("who talks to whom"). The authors exploit network-wide interactions between hosts for extracting graph structures from network traffic datasets, considering each node as a distinct IP address and edges as their communication flows. Although the definition of the TDG's nodes is a simple process, the principal task is the definition of the edges. This can be done based on the available information as, for example, the first sent packet, the amount of exchanged information, the protocol used and so on. These edges can be both directed or undirected, according to the final goal. This mapping represents a viable solution when inputs are network traffic data. In particular, it allows monitoring, analyzing and visualizing relations among defined nodes using social interaction paradigms (Iliofotou et al., 2007). In Iliofotou et al. (2009a), the concept of TDGs is extended introducing a temporal factor. The authors propose to split the initial data into subintervals and extract a graph from each subinterval.

In this study, temporal TDGs are used as a starting point for translating the attack classification problem into a node behaviour classification task.

2.2.2. Class Imbalance Problem

One of the most common challenges when facing classification tasks is addressing an uneven distribution among classes, which can produce skewed behavioural models (Fernández et al., 2018). In this regard, data-level techniques are one of the most straightforward and effective ways to address class imbalance (Leevy et al., 2018). These techniques can be divided into under-sampling, over-sampling or hybrid methods, depending on how they modify the data distribution (removing or adding samples from the dataset).

The simplest, yet effective alternative for under-sampling is known as Random Under-Sampling (RUS), removing random samples from the majority class until the dataset is balanced. Other methods in this category are *Tomek links* (Tomek et al., 1976), *Condensed nearest neighbor rule* (Hart, 1968) and *Near miss undersampling* (Yen and Lee, 2006). Among the over-sampling approaches, the most common are *Random Over-Sampling (ROS)* and *Synthetic Minority Oversampling Technique (SMOTE) (Chawla et al., 2002)*. ROS takes samples from less represented classes and randomly replicates them until reaching the population size of the majority one, whereas SMOTE is based on exploiting k-nearest neighbors to create new elements. Other over-sampling methods are *Adaptive Synthetic (ADASYN)* He et al. (2008) and *Bordeline oversampling* Nguyen et al. (2011).

When dealing with graph structured data, the application of these techniques is not straightforward and can be misleading. In fact, these techniques are focused on the feature space and if applied without modifications the graph structure might be modified, changing its intrinsic information. For this reason, several techniques have been proposed for working with graph structures in order to reduce their size and preserve properties, as shown in Hu and Lau (2013). Commonly studied sampling techniques can be grouped in three classes (Wu et al., 2016): *Node-based samplings, Edge-based samplings* and *Transversal-based samplings*. Random Node technique Leskovec and Faloutsos (2006) and Random Degree Node Stumpf et al. (2005) are typical examples of the first strategy; Random Edge Technique, DropEdge Rong et al. (2019) and GAUG Zhao et al. (2020c) belongs to *Edge-based samplings*. Finally, Snowball Sampling Stivala et al. (2016) and Forest Fire Sampling (FFS) Leskovec and Faloutsos (2006) are *Transversal-based samplings* strategies.

All introduced approaches, although with different impact, generate changes in the graph topology, thereby altering its intrinsic information. For this reason, we present two novel approaches for handling graph data-level preprocessing based on the idea of preserving subgraph properties and topology (Section 3.3).

2.2.3. Graph Convolutional Network

In recent years, Deep Learning has been successfully applied in many fields related to image, video, recorded speech data etc. (Garcia-Garcia et al., 2018). However, when real world applications are characterized by complex relationships and interdependence between samples, new learning paradigms are required to exploit such kind of structures (Wu et al., 2020). Graph Neural Networks (GNNs) were introduced in Scarselli et al. (2008) to handle supervised machine learning tasks using graph structured data (cyclic, directed, undirected, or a mixture of these) in which each node is defined by its features and its relation to other nodes in the graph. The idea of GNNs is to generate outputs by analyzing the embedded state of a node, which also contains information about the neighborhood of the node itself (Zhou et al., 2018). Further, Graph Convolutional Networks (GCNs) (Kipf and Welling, 2017) exploit the local and global structural patterns of a graph. The aim of GCNs

is to consider the relation among nodes through a convolution operation similar to the one used in Convolutional Neural Networks (CNNs). While CNNs and GCNs have very similar underlying key concepts, CNNs are specifically built to operate on Euclidean data, while GCNs are suited for non-Euclidean data such as graphs that contain nodes, connections, relations and unordered information. According to the chosen filter and its application, graph convolution operations can be separated into spatial-based and spectral-based convolution (Zhang et al., 2019). In the first category, the operation is applied directly on the graph nodes and its neighborhood, i.e. through some aggregations of graph signals within the node neighborhood. In the second case, the filters are defined following graph signal processing concepts, i.e. using graph Laplacian Matrix to define the graph Fourier transformation, and the graph filtering operators within (Defferrard et al., 2016; Kipf and Welling, 2017).

While it is straightforward to compute the convolution in the spatial domain (for example by applying a weighted average function over a node and its neighborhood (Hamilton et al., 2017; Monti et al., 2017)), it is more complex in the spectral domain. In particular, in this second case, the convolution operation is defined through Equation 1, where $x \in R^N$ represents a scalar vector (a scalar for every node in the graph), U is the matrix of eigenvectors of L, the Laplacian of the graph, and $g_\theta(\Lambda)$ is a function of the eigenvalues of L and it represents the filter in the Fourier domain. Nevertheless, solving this equation can be computationally complex and unreachable, particularly for large graphs.

$$y = g_\theta * x = U g_\theta(\Lambda) U^T x \tag{1}$$

For this reason, a solution for Equation 1 is obtained by parameterizing the term $g_\theta(\Lambda)$ as a polynomial function that can be computed recursively. More specifically, Chebyshev polynomials with a K degree can be used, as shown in Defferrard et al. (2016) (Equation 2). In particular, $g_\theta(\Lambda)$ can be approximated by a truncated expansion of the Chebyshev polynomials $T_k(x)$ up to K degree, where $T_k(x)$ is recursively defined as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$. In Defferrard et al. (2016), the authors suggest rescaling the filters by $\widetilde{\Lambda} = \frac{2}{\lambda_{max}} \Lambda - I_N$, where λ_{max} is the largest eigenvalue of Laplacian matrix L and I_N is the identity matrix. $\theta' \in \mathbb{R}^K$ is a vector of Chebyshev coefficients.

$$g_{\theta'}(\Lambda) \simeq \sum_{k=0}^{K} \theta'_k T_k(\widetilde{\Lambda}) \tag{2}$$

Kipf et al. Kipf and Welling (2017) demonstrate that a good approximation can be reached by truncating the Chebichev polynomial to get a linear polynomial ($k = 1$) and by performing a renormalization trick in order to avoid numerical instabilities and vanishing gradients. In this scenario, Equation 1 is reduced to Equation 3 (Kipf and Welling, 2017), where σ represents an activation function, e.g., $ReLU(\cdot) = \max(0, \cdot)$, $H^{(l)}$ is the matrix of activations in the l^{th} layer with $H^{(0)} = X \in \mathbb{R}^{NxC}$, i.e. a matrix of N nodes each one associated with a C-dimensional feature vector, and $W^{(l)}$ denotes a layer-specific trainable weight matrix. The term $\widetilde{A}$ is the adjacency matrix with the identity matrix added (part of the renormalization trick), and $\widetilde{D}_{ij} = \sum \widetilde{A}_{ij}$. Basically, the term $\widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}}$ is used to normalize the graph structure and to convert it to a regular neural network function. In Kipf and Welling (2017), an in-depth discussion of this approximation is presented.

$$H^{(l+1)} = \sigma(\widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}) \tag{3}$$

Thanks to its ability to learn graph representations, Graph Convolutional Networks are used in a wide range of applications, especially for detecting similarity among networks and for discovering patterns among the nodes' relations. Examples include applications in chemistry, biology and bioinformatics for classifying drugs (Long et al., 2020), chemical reactions (Coley et al., 2019), molecules (You et al., 2018) and material properties (Xie and Grossman, 2018), or in social science for implementing recommendation systems (Jin et al., 2020; Wu et al., 2018). Regarding the cybersecurity domain, Zhao et al. Zhao et al. (2020a,b) first used a GCN to transform a botnet detection problem into a semi-supervised classification problem and proposed a new framework for cyber threat discovery based on multi-granular attention and Indicator of Compromises (IOCs) extraction. In Gao et al. (2021), heterogeneous graphs and GCNs are combined for classifying Android malware, meanwhile, for the same task, in Pei et al. (2020), the GCN and recurrent networks are used for identifying and learning semantic and sequential patterns. Oba et al. Oba and Taniguchi (2020) presented a solution based on a GCN able to analyze a multigraph based on triplets of client IP, server IP and TCP/UDP ports. Crovella et al. Crovella and Kolaczyk (2003) use graphs for representing and analyzing incoming and outgoing flows of an access link and identify denial of service (DoS) attacks. In Nagaraja et al. (2010) and in Wang et al. (2020) graph-based information is combined with flow-based data for detecting botnets.

In Sun et al. (2020b) and Sun et al. (2020a), new malicious domain detection systems based on GCNs were introduced exploiting Heterogeneous Information Network for analyzing relations among domains, clients and IP address and finally extract meta-path information for detecting malicious activities. Jiang et al. Jiang et al. (2019) presented a GCN-based anomaly detection system for threat and fraud detection. Results showed that GCNs outperform 4 state-of-the-art techniques. In Zheng et al. (2019), an anomalous edge detection framework based on GCNs with an attention model (Gated Recurrent Unit) was presented.

Following these promising trends, we explore here benefits and limitations of graph convolutional approaches for node behaviour classification and compare them with unsupervised anomaly detection techniques, as well as with a more traditional neural network approach, where only feature information is used.

2.2.4. Anomaly detection (AD)

Anomaly detection (AD) is a task in which models learn the distribution of given data and try to detect points that are different from the norm, thereby classifying them as anomalies (Chandola et al., 2009). Even though this paper is focused on presenting a novel graph-based methodology to address graph-related class imbalance to improve *node behaviour* classification using supervised approaches, it is interesting to compare our results with traditional anomaly detection approaches that can be directly applied to the imbalanced graph-based dataset. For this reason, as part of our study, we implemented 5 different anomaly detection models, which, although all based on unsupervised machine learning, can be separated into *semi-supervised AD* and *unsupervised AD* based on their training setup (Aggarwal, 2017),(Goldstein and Uchida, 2016) (Section 4.5). This selection of algorithms is not exhaustive but we think that it represents a good benchmark set for evaluating the benefits and limitations of our approach. In particular, *Isolation Forest* (IForest) (Liu et al., 2012), *Local Outlier Factor* (LOF) (Breunig et al., 2000), *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN) (Ester et al., 1996), *k-Nearest Neighbors* (*k*-NN)(Liao and Vemuri, 2002) and *Autoencoder* (AE) (Zhou and Paffenroth, 2017) were implemented here.

3. Methodology

In this section, a novel methodology is presented, which converts network traffic classification into node behaviour classification. In Section 3.1, the problems to be addressed and the main contributions of this work are presented. In Section 3.2 and

Section 3.3, the operations for extracting the Traffic Dispersion Graphs (TDGs) via temporal dissection and the novel graph data-level preprocessing techniques are described. Finally, in Section 3.4, the applied classification operations are detailed.

3.1. Motivation and Contributions

In graph analysis based on real use cases, it may be useful to not only classify a single node's communication, but also to detect the entity that has generated the communication in order to isolate or exclude it from the network. However, when graph structures are extracted from network traffic data, it should be noted that it is not possible to use the whole dataset at once for creating a unique static and monolithic graph. In fact, due to the structure of common network traffic datasets that usually involve captures of many hours or even days, the amount of data in the created graph can require too much computational effort, making the application of these graph approaches difficult. For example, using a graph generated by a 10-hour long capture-day for training a model, the final application (the test), would require a graph of comparable size (other $\sim$ 10 hours). This can compromise the usability of the trained model as attack detection timing is fundamental in order to mitigate a threat promptly.

For this reason, in this work, we propose to extract and analyze graph-based information from time series data (TDGs) by defining entities/nodes and their links/edges. Furthermore, we add an operation of temporal dissection, i.e. a fragmentation of the initial dataset into fixed time intervals (or *temporal snapshots*) to highlight network micro-dynamics and hence increase the usability of the solution. In particular, a study analyzing the effect of three different temporal snapshot sizes on the behaviour definition is presented. Further, an enrichment operation is performed in which graph features are extracted from each TDG and added to the entities, enhancing their behaviour description.

As these temporal TDGs show a strong graph class imbalance (due to the dominance of traffic information related to normal activities rather than attack connections), we analyze and compare two different approaches. On the one hand, we test unsupervised models for anomaly detection, which directly assess the imbalanced dataset. On the other hand, we implement two supervised machine learning approaches for node behaviour classification; one that involves behavioural descriptions only (Neural Networks or NN) and two others that additionally use graph neighborhood information (Graph Convolutional Networks or GCN). Specifically, two distinct approximations for the GCN convolution filter are tested. Furthermore, in order to improve the supervised machine learning results, two novel techniques for addressing the class imbalance problem in graph-structured data are introduced. These two graph data-level preprocessing techniques called *R-hybrid* and *SM-hybrid*, exploit the fragmented TDGs for reducing their impact on subgraph topology.

Figure 1 shows how the methodology for the supervised ML is separated into three distinct phases: *Graph creation; Data-level Preprocessing;* and finally, *Classification*. Note that the unsupervised approach does not need the *Data-level Preprocessing* phase.

In summary, the main contributions of this work are:

- A temporal analysis using different time intervals for transforming network traffic data into graph-based structures (TDGs) is presented;
- Connection information is converted into node behaviours for characterizing graph entities;
- Two novel graph data-level preprocessing techniques are introduced to tackle class imbalance in graph data;
- Three Deep Learning approaches are compared in terms of node behaviour classification performance (one based on behavioural features only and the other two using graph relational information as well).
- The results obtained using our approach based on supervised machine learning and based on a balanced dataset (*classification*) are compared to the ones obtained using 5 state-of-the-art unsupervised techniques applied to the imbalanced dataset (*anomaly detection*).

3.2. Phase 1: Graph creation

The first phase can be split into two parts; the first one focussing on the temporal dissection and TDG creation, and the second one implementing an enrichment process.

3.2.1. Temporal Dissection and Traffic Dispersion Graphs (TDGs)

As the node behaviour classification task is here addressed as a supervised machine learning task, an initial labelled dataset is required, which is relevant for the definition of normal and attack behaviour. Figure 2a shows how the time series datasets contain information about traffic flows between a source and a destination, each one identified by several features such as IP, port, protocol, bytes and so on. We propose to use a temporal dissection operation, which allows us to split the network traffic dataset into fixed-time intervals, generating so-called temporal snapshots. From each of these, Traffic Dispersion Graphs (TDGs) are extracted.

Each TDG is characterized by *nodes* - also referred to as *entities* - as a combination of IP and port number, and *edges* that indicate if traffic is exchanged between nodes. Following these definitions, each row of the network traffic dataset can be seen as an undirected edge. In fact, each row contains information about data exchanged between the source and the destination, as well as the information about the response sent by the destination to the source (bidirectional flow). In this way, each row can be reduced to a 4-tuple of source entity, destination entity, edge features and edge label (Figure 2b). This operation allows us to generate a first version of the TDG from each temporal snapshot, as shown in Figure 2c. However, traffic information is still stored on the edges of the graph, which is why an operation for translating them into node behaviour is required (Figure 2d). This operation is performed by combining all the edge feature vectors in which a certain node is involved. Let all the edges related to node$_i$ be described as a feature vector $e_j = \{f_{j_1}, f_{j_2}, ..., f_{j_N}\}$, where N is the number of features and f_{j_h} represents the $h-th$ feature of the $j-th$ edge, then it is possible to compute node$_i$'s behaviour B_i by combining its edge features with an averaging operation. For example, following Figure 2d, B_3, i.e. the behaviour of the *node*3, could be computed by combining the edges' features vectors of e_3, e_7 and e_8 as indicated in Equation 4.

$$B_3 = \{\frac{f_{3_1} + f_{7_1} + f_{8_1}}{3}, \frac{f_{3_2} + f_{7_2} + f_{8_2}}{3}, ..., \frac{f_{3_N} + f_{7_N} + f_{8_N}}{3}\} \quad (4)$$

Furthermore, it is possible to enhance the node behaviour description by enriching it with m additional features that can be computed considering extra information such as the number of joined edges, the maximum number of edges with the same entity, and so on. This operation increases the behavioural vector dimensionality from N to $N+m$ elements.

Once TDGs are generated and edge information is converted into node behaviours, it is important to assign a label to each node in order to distinguish normal and attacker nodes. For executing common attacks like Reconnaissance, Denial-of-Service, Port Scan, Exploits, Fuzzers and many others in real use cases, attackers represent active entities that start the communication with its targets. Hence, in order to label normal and attacker behaviour, only labels related to edges in which a node appears as source entity are considered and combined. If among these labels the majority are nor-

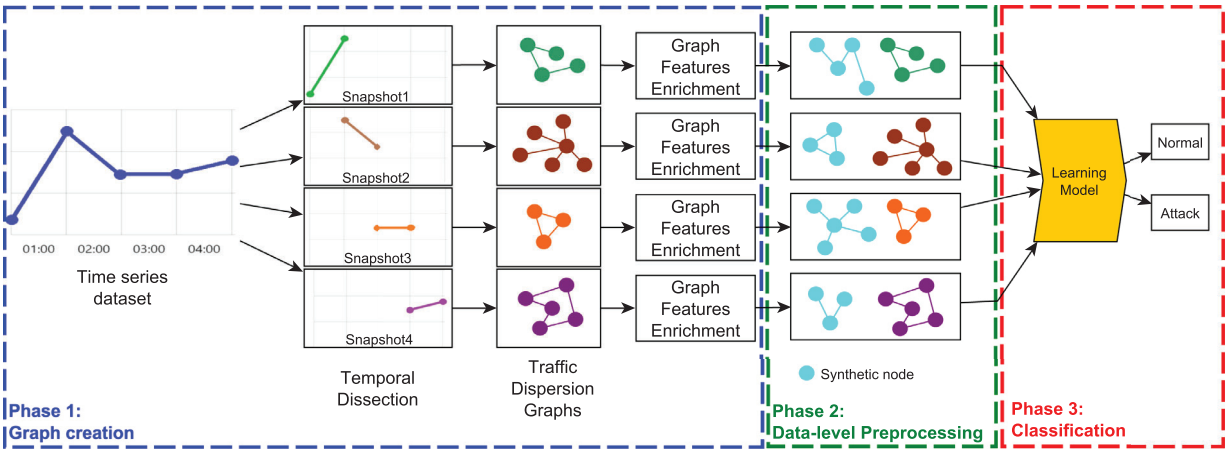


Fig. 1. Methodology phases.

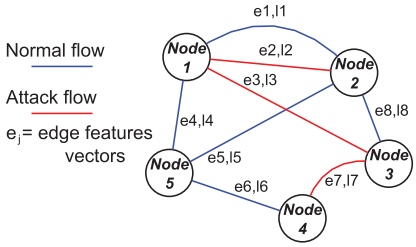
source IP	source PORT	destination IP	destination PORT	protocol	timestamp	source bytes	destination bytes	...	label
A.A.A.A	1234	B.B.B.B	4321	TCP	1421987377	1000	1000	...	0
A.A.A.A	1234	B.B.B.B	4321	UDP	1421987378	80	0	...	1
C.C.C.C	5678	A.A.A.A	1234	TCP	1421987379	176	245	...	1
...	...	...	...	...	...	...	...	...	...

(a) Example of network traffic dataset structure

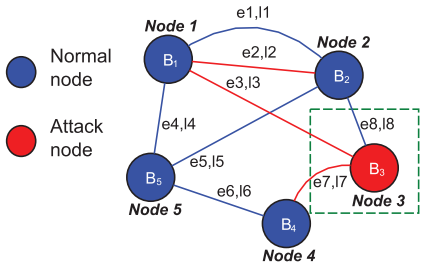
source entity
destination entity
edge features
edge label

source entity	destination entity	edges	labels
Node 1	Node 2	e1	l1
Node 1	Node 2	e2	l2
Node 3	Node 1	e3	l3
...	...	...	...

(b) Example of extracted triplets



(c) TDG with features on edges



(d) TDG in which node behaviours are computed combining edges features vectors

Fig. 2. Pipeline for extracting Traffic Dispersion Graphs (TDGs).

mal edges, the node is labelled as normal. Otherwise, it is labelled as attacker.

From here onward, in this paper, the terms *behavioural features* will refer to the features that define each node behaviour, and *0-class* and *1-class* will be used to indicate *normal* (0) and *attacker*(1) behaviour, respectively.

3.2.2. Graph Features Enrichment

Our approach proposes to extract new information from TDG structures in order to enhance/enrich node descriptions. In this manner, each node behaviour is ultimately identified by both behavioural and graph features.

Creating the temporal TDG based on the network traffic dataset and defining nodes as a combination of IP and port number generates highly fragmented graphs characterized by many subgraph structures (Figure 3), which is in accordance with literature (Iliofotou et al., 2009a). In fact, it is normal that in a fixed, short time interval activities related to a certain node are limited - which promotes the creation of simple and disconnected graph-structures. Yet, in rare cases, complex structures are generated in which the subgraph is composed of two dense parts that are only connected by one node. In order to denote these complex structures, the concepts of *r-nodebridge, set of r-nodebridges* and *e-bridge* are defined.

Definition 3.1. Let $G = (V, E)$ be a graph and $u, v, z \in V$ be nodes of G with $u \neq v \neq z$. Let $e_1 = \{v, u\}, e_2 = \{v, z\} \in E$ be the only two adjacent edges between those nodes, and suppose that both e_1 and e_2 are bridges of G.

Let $G'_{e_i} = (V'_{e_i}, E'_{e_i})$ and $G''_{e_i} = (V''_{e_i}, E''_{e_i})$ with $i = 1, 2$ be the components obtained by cutting e_1 or e_2, respectively and let $r = \max_{i=1,2} \min\{|V'_{e_i}|, |V''_{e_i}|\}$. Then, v is called *r-nodebridge* and W_s is the *set of r-nodebridges* for $r \geq s$. The removed edge will be called *e-bridge*.

This definition allows for detecting all elements of the W_s. For each one, its *e*-bridge is removed in order to split the two dense parts and create two distinct subgraphs, thereby reducing the overall complexity.

In order to improve the definition of node behaviour, several graph metrics are directly extracted from the temporal snapshot and added to each node behaviour. In particular, 12 common graph metrics are chosen, as indicated in Table 1. Three of them are first computed by considering the whole graph in a temporal snapshot

Table 1
Overview of graph metrics used as features.

#	Graph features	Category	Input	Range	Description
1	Degree Centrality	Centrality	Graph Subgraph	[0, 1]	represents the normalized degree of a node, i.e. dividing it by the maximum possible degree of the considered graph (or subgraph).
2	Betweenness	Centrality	Graph Subgraph	[0, 1]	quantifies the number of times a node acts as a bridge along the shortest path between two other nodes, divided by the total number of the shortest paths between them.
3	Closeness	Centrality	Graph Subgraph	[0, 1]	represents the reciprocal of the average length of the shortest path between the node and all other nodes in the input graph (or subgraph). Higher values of closeness indicate higher centrality.
4	Second Order Centrality	Centrality	Subgraph	[0, +inf)	is the standard the deviation of the return times of a permanent unbiased random walk running on the topology Kermarrec et al. (2011). Lower values of second order centrality indicate higher centrality.
5	Edge Betweenness	Centrality	Subgraph	[0, 1]	represents the betweenness centrality of an edge e (the same concept as defined in Betweenness, but applied to an edge). In our case, edges centrality that involve the same node are summed.
6	Eccentricity	Distance	Subgraph	[0, +inf)	is the maximum distance of a node to all others in the considered subgraph.
7	Barycenter	Distance	Subgraph	0 or 1	indicates if the node is part of the barycenter of the considered subgraph.
8	Center	Distance	Subgraph	0 or 1	indicates if the eccentricity of a node is equal to its subgraph radius.
9	Radius	Distance	Subgraph	[0, +inf)	is the minimum eccentricity in each considered subgraph
10	Size	Shape	Subgraph	[1, +inf)	is the number of nodes in the subgraph to which the considered node belongs.
11	Weighted Degree	Shape	Node	[0, +inf)	is the sum of weights of all connections in which a node is involved.
12	Degree two hops	Relational	Node	[0, +inf)	represents the number of connections of a node considering two hops, i.e. two consecutive connections.

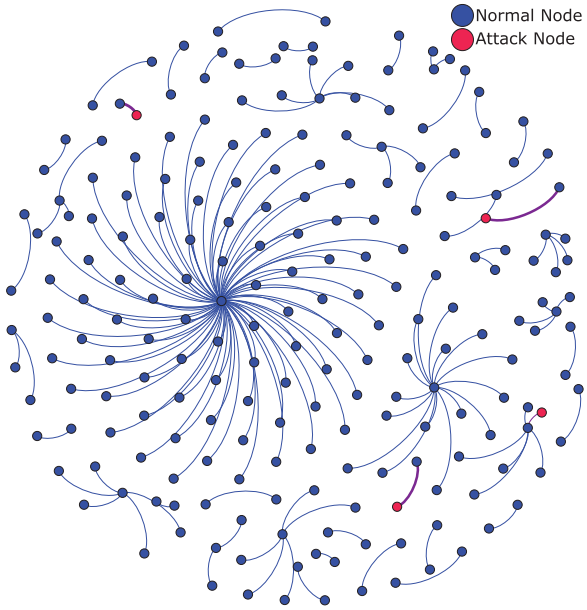


Fig. 3. Example of a graph extracted from a temporal snapshot (reduced).

and then by considering the concrete subgraph to which the node belongs, obtaining a total of 15 graph features.

3.3. Phase 2: Data-level preprocessing

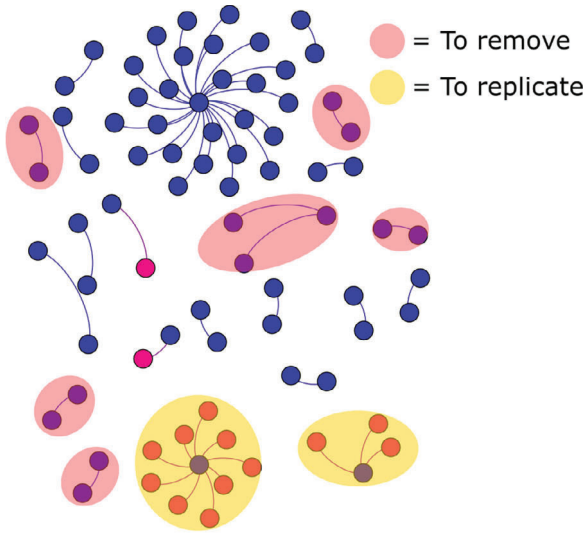
While the process described in Section 3.2 transforms traffic data into graph behavioural information, it does not address the class imbalance problem. The distribution of normal and attack behavioural nodes generated in Phase 1 keeps suffering from the same imbalance as the initial dataset.

For this reason and for improving the supervised ML results, in Phase 2, the idea is to balance the number of attackers and normal nodes introducing two novel techniques that are able to create new synthetic samples for each temporal TDG. On the one hand, these operations try to remove repeated subgraphs in which normal nodes (majority class) are involved. On the other hand, they try to replicate the most "interesting" subgraphs in which the local-majority population is made up of attacker nodes (minority class). In this way, these methods address the graph imbalance problem without modifying the structural topologies of each TDGs' subgraphs (Figure 4). More specifically, the first approach combines RUS and ROS techniques to obtain a comparable distribution among the classes for each TDG (named as *R-hybrid*), whereas the second approach exploits RUS, SMOTE and ROS techniques (named as *SM-hybrid*).

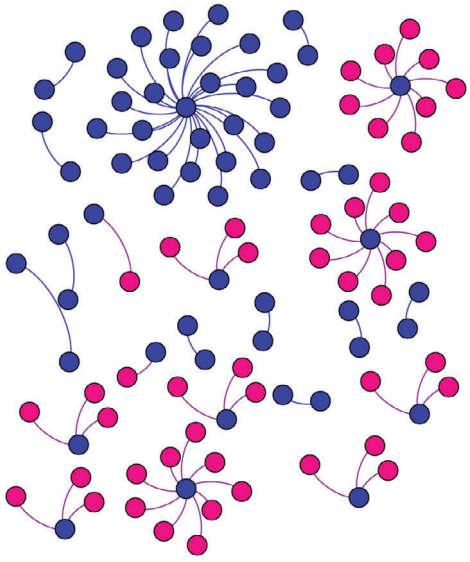
As mentioned, the first step is the same for both approaches - detecting subgraph structures in each temporal snapshot that are characterized by nodes belonging to the most represented class only (normal behaviour). Once these structures are detected, they are randomly removed for each snapshot until the population of that class is halved (Figure 4a). In this manner, direct relational information is not modified and neither are the structures of the untouched subgraphs.

For each TDG subgraph, structures that are characterized by at least 60% of nodes belonging to the less represented class (attacker behaviour) are then selected and randomly replicated, as shown in Figure 4b. The chosen percentage between attacker and normal nodes in the selected subgraphs is justified for ensuring the convergence of this approach, i.e. in each iteration there are more elements added belonging to the less represented class than elements belonging to the majority class, allowing to reach a balanced population. The replication process is stopped when the number of attacker nodes reaches the initial population of the normal nodes, i.e. the population before starting the replication operation. Although both approaches replicate the subgraph structures in the same way, their main difference is related to how they replicate single node behaviour in each subgraph, as shown in Figure 5.

1. *R-hybrid*: after applying RUS, this technique replicates not only the most relevant subgraphs, but also their node behaviours using a ROS strategy, as shown in Figure 5a.
2. *SM-hybrid*: after applying RUS, this technique replicates the most relevant subgraphs, however, normal behaviour are replicated using ROS technique, whereas attacker nodes are gen-

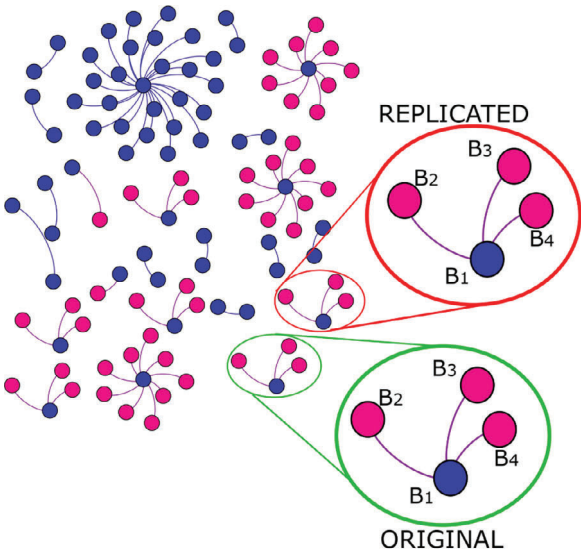


(a) Detection of structures to remove (normal nodes) and to replicate (under-represented attacker nodes)

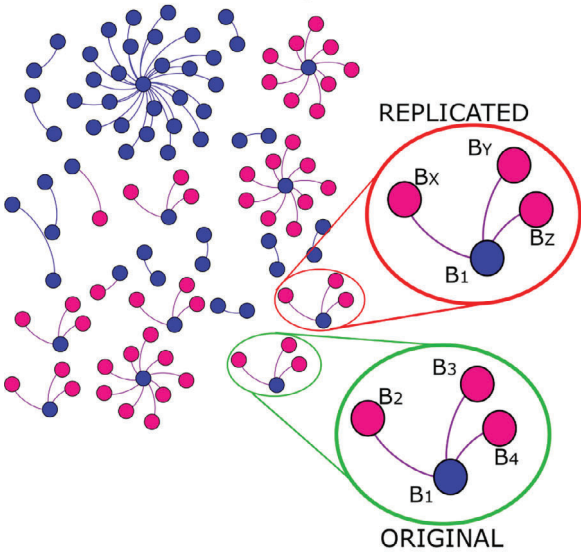


(b) Example of the augmented graph, classes are more balanced

Fig. 4. Example of the proposed method for data-level augmentation on graph data.



(a) R-hybrid



(b) SM-hybrid

Fig. 5. Difference in node behaviour generation using R-hybrid and SM-hybrid methods.

erated by applying a SMOTE technique considering the behavioural features of the whole attacker nodes only (B_x, B_y and B_z in Figure 5b). In fact, graph features are not considered in the SMOTE process since they directly depend on the chosen subgraph and can be calculated afterwards, as explained in Section 3.2.2. In order to apply the SMOTE technique, the minority class should be characterized by a minimum number of neighbors (N). When this condition is not satisfied, the ROS technique is used instead.

3.4. Phase 3: Classification

Finally, in Phase 3, the node behaviour classification is performed by training Deep Learning models. Considering the presented data structures, we opted for comparing two distinct learning approaches - Neural Networks (NN) using behavioural and graph features to classify normal and attacker nodes and Graph Convolutional Networks (GCNs), which combine neighborhood information extracted from the graph via a convolution operation. Here, we investigate two GCN implementations where Chebychev polynomials up to K degrees are used for modelling the (spectral) convolutional filters, as described in Section 2.2.3. In particular, the first implementation is based on maximum Chebychev degree K equal to 3 (ChebyNet or 3-GCN), and the second implementation based on a Chebychev simplification (linear polynomial) presented in Kipf and Welling (2017), called first-order GCN or 1-GCN.

4. Experimental Framework

A network traffic dataset is used to validate the methodology introduced in this work. In Section 4.1 and Section 4.2, the dataset as well as its limitations are described. In Section 4.3, an overview of the temporal dissection and the graph results are presented, whereas Section 4.4 and Section 4.5 show the metrics used to evaluate the Deep Learning models and their settings. Finally, the experiments are detailed in Section 4.6

4.1. Dataset Overview and Feature Selection

To test our approach for node behaviour classification, a widely used cybersecurity dataset (Monshizadeh et al., 2019; Zhang et al., 2018) called UNSW-NB15[6] is used. This dataset was created with the aim to improve existing benchmark datasets, which may not be able to provide a comprehensive representation of network traffic and attack scenarios (Moustafa and Slay, 2015). The UNSW-NB15 dataset contains real normal and synthetic abnormal network traffic generated in the University of New South Wales (UNSW) cybersecurity lab. UNSW-NB15 is characterized by nine major families of attacks (Fuzzers, Analysis, Backdoors, Denial of Services, Exploits, Generic, Reconnaissance, Shellcode and Worms) and normal traffic generated over two distinct capture days, the first with a simulation period of 16 hours and the second with a simulation period of 15 hours. Although it is not always possible to clearly distinguish attack families due to their multi-purpose applications and correlations, in this study the following three categories are considered in order to facilitate the description of the nine families: attacks for gathering information, attacks for making a resource unavailable and attacks for taking control of a target (hijacking):

Gather information

1. *Analysis:* composed of different attacks like port scan, spam and HTML file penetrations.
2. *Generic:* techniques that try to decrypt information using methods that work against block ciphers (with a given block and key size).
3. *Reconnaissance (or Gather):* attacks used for discovering more information about a target, usually to start an investigation before deploying the real attack.

Target unavailable

4. *Fuzzers:* attacks that randomly send data as input to a target in order to exploit vulnerabilities and bugs for generating failure, crashes or unwanted behaviour.
5. *Denial-of-Service (DoS):* method that tries to make a target unavailable by flooding it with traffic, or by sending information that triggers a crash.

Hijacking

6. *Backdoor:* technique that uses methods for gaining authorized and/or unauthorized access bypassing system security mechanisms.
7. *Exploits:* attacks that take advantage of known weaknesses of the target (bugs or vulnerabilities) to take over control and/or steal information.
8. *Shellcode:* method that uses a small piece of code (usually command shell) to exploit software vulnerabilities and take control of the compromised machine.
9. *Worms:* malware that has the ability to self-replicate and spread automatically across multiple devices exploiting target vulnerabilities.

Tools like Argus[7] and Bro-IDS[8] are used to generate a first preprocessed dataset, in which network packets are aggregated in cumulative records - each one defined by a total of 49 features, including two different class labels generated by the detection tools. One general class label indicates if a connection represents a normal activity (0) or an attack (1), whereas the second label specifies the attack category according to the nine available categories. This labelled dataset is available in a CSV format, and contains clean data without missing values or duplicated records (Moustafa, 2017). Using this labelled version of the UNSW-NB15 dataset reduces the captured duration to $12h : 35m : 10s$ and $11h : 57m : 41s$ for the first and second capture day, respectively. This preprocessed dataset is characterized by 2,540,044 labelled flows of which 2,218,761 are labelled as normal and 321,283 are labelled as attack flows. As described in other studies (Moustafa and Slay, 2017; Zhang et al., 2018), not all of the 49 available features are relevant for the classification. Hence, only the following 21 features were used and combined here for defining and extracting node behaviours: *source/destination IP, source/destination port number, transaction protocol, state protocol, record total duration, source/destination to destination/source bytes, source /destination to destination/ source time to live, source/ destination packets dropped, source/ destination bits per second, services (dns, ftp, ssh,..), source/ destination to destination/ source packet count, record start timestamp, record label and category label.*

4.2. Dataset limitations and clean up

As presented in Section 4.1, nine attack families are given in the UNSW-NB15 dataset. However, the dataset suffers from two major problems: *class imbalance* and *class overlap* (Zoghi, 2020). In this study, class imbalance is addressed by applying novel data-level preprocessing algorithms in the graph domain, whereas class overlap is mitigated by reducing noisy information and by focusing the analysis on the most populated protocols and services only.

Class overlap is created when the space generated by class features from one class overlaps the space generated by features from another class. As shown in Zoghi (2020), several classes of the UNSW-NB15 dataset suffer from class overlap. To mitigate the problem and reduce the randomness of the data without losing important information, the analysis was focused on particular services and protocols only, following previous suggestions (Zhang et al., 2018). More specifically, only services identified as *undefined, dns, http, smtp* and *ftp* were chosen. This filtered dataset was characterized by more than 100 protocols. As not all them have the same relevance, we decided to keep only the 16 most represented protocols. The final filtered dataset loses less than 10% of the originally available information, as shown in Table 2, where the size of the dataset for each category is shown before and after filtering. Regarding the population of the 1-class (attack, malicious class), the *Worm* family was excluded from the analysis due to its low representation (0.06%), reducing the considered attack families to eight.

Even though the main goal of this study is to compute a binary classification to distinguish between normal and attacker behaviour, the eight remaining attack families are considered individually during the data-level preprocessing operations. In this manner, all considered attack subcategories are replicated often enough, mitigating the creation of *small disjuncts* in the classification problem. A small disjunct is a disjunct that covers only a few training examples and that generates high error rates in testing (Weiss, 2010).

To the best of our knowledge, this is the first time that this dataset has been used for defining a structured graph to highlight and classify node behaviours.

4.3. Temporal graph extraction

As explained in Section 3.1, we need to extract temporal TDGs (graphs) from the network traffic dataset (time series) prior to running experiments. The first operation is to fragment the dataset into fixed time intervals (temporal dissection). Here, three tempo-

[6] https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/
[7] https://qosient.com/argus/index.shtml
[8] https://zeek.org/

Table 2
Comparison between class distributions in the initial and the filtered, cleaned dataset.

#	Category	Label	# Initial dataset	# Filtered dataset	% Population 1-class
0	Normal	0-class	2,218,764	2,037,045	-
1	Analysis	1-class	2,677	1,773	0,59
2	Generic	1-class	215,481	214,616	71,09
3	Reconnaissance	1-class	13,987	13,116	4,34
4	Fuzzers	1-class	24,246	23,116	7,66
5	DoS	1-class	16,353	11,204	3,71
6	Backdoor	1-class	2,329	1,527	0,51
7	Exploits	1-class	44,525	34,841	11,54
8	Shellcode	1-class	1,511	1,511	0,50
9	Worm	1-class	174	174	0,06
	total		**2,540,047**	**2,338,923**	**100**

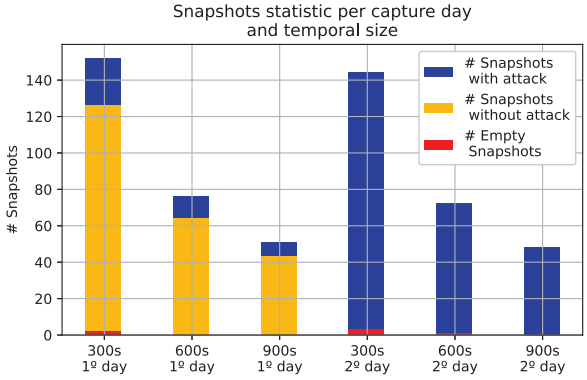


Fig. 6. Overview of temporal configurations and created temporal snapshots for both capture days.

Table 3
Overview of identified nodes per class in the second capture day.

#	Category	300s	600s	900s
0	Normal	1,181,488	1,166,477	1,158,271
1	Analysis	358	341	338
2	Generic	3,256	3,174	3,149
3	Reconnaissance	9,862	9,831	9,810
4	Fuzzers	15,811	15,667	15,649
5	DoS	2,988	2,973	2,963
6	Backdoor	296	295	294
7	Exploits	19,606	19,526	19,482
8	Shellcode	1,290	1,288	1,289
	total	**1,234,955**	**1,219,572**	**1,211,245**
overall population 0-class		**95.47%**	**95.45%**	**95.43%**
overall population 1-class		**4.53%**	**4.55%**	**4.57%**

ral snapshots sizes are chosen: 300*s*, 600*s* and 900*s* and for each configuration, TDGs are built.

In Figure 6, the number of snapshots obtained in each configuration depending on the capture day is reported, as well as the number of snapshots that are fully characterized by normal nodes (without attacks) and the number of snapshots with an empty population (without any kind of nodes). Temporal dissection generates different results depending on the available data and the duration of the capture day. In particular, more than 80% of the created snapshots in the first capture day do not show attacker behaviour. For this reason, only non-empty snapshots obtained from the *second* capture day are considered - which are 141, 71 and 48 for 300*s*, 600*s* and 900*s*, respectively.

Table 3 shows that the overall number of 1-class nodes is still much lower than the number of nodes belonging to the 0-class when considering all nodes in all extracted TDGs on the second capture day. This once again highlights the strong class imbalance that affects this dataset (imbalance rate $\sim$ 22). In fact, the 1-class population represents less than 5% of the whole graph-based information regardless of the chosen temporal snapshot size. Furthermore, a strong reduction of the *Generic* family is shown after the Graph creation phase even though it was the most represented family in terms of single connections (Table 2). This is because there are only a few distinct nodes that belong to this family (Table 3) that generate large attack traffic (flows).

Once these temporal TDGs are extracted, the elements of the set W_s of each TDG are detected and all *e*-bridges associated with them are removed in order to simplify rare enclosed complex graph structures. In particular, to detect the elements of the W_s set, we chose *s* equal to 6. After this reduction, graph features are computed and used for enriching the node behaviours description, generating the inputs for the experiments.

4.4. Evaluation Metrics

Classification metrics used to evaluate the graph Deep Learning model implemented in our experiments are obtained via the confusion matrix related to binary classifications, defining the values of True Positives, True Negatives, False Positives and False Negatives.

- True Positives (tp) represent cases in which both the predicted and the real label indicate anomalies/attacks (class 1).
- True Negatives (tn) represent cases in which both the predicted and the real label indicate no-anomalies/normal (class 0).
- False Positives (fp) represent cases in which the model predicts anomalies (class 1), however the real label indicates no-anomalies (class 0).
- False Negatives (fn) represent cases in which the model predicts no-anomalies (class 0), however the real label indicates anomalies (class 1).

Starting from the confusion matrix, 5 important metrics are calculated for evaluating the learning classifier: *Accuracy, Precision, Sensitivity, False Prositive Rate (FPR)* and *F1-score*. *Accuracy* represents an overall effectiveness of a classifier; *Precision* is a measure of a classifier's exactness; *Sensitivity* represents a measure of a classifier's completeness; *FPR* indicates the ratio of negative elements predicted as positive ones; *F1-score* shows the relation between actual positive labels and those given by the classifier. These metrics, as well as their formulas and range values are reported in Table 4. Furthermore, we include the *Area Under the Receiver Operating Characteristic Curve (AUC-ROC or AUROC)* (Fawcett, 2006) for evaluating and comparing different classification performances. The *AUC-ROC* represents the classifier's ability to distinguish between classes and it takes values in a range between 0 and 1.

To perform the experiments, for the supervised models, the temporal TDGs are randomly separated into train, validation and test datasets keeping a fixed proportion of temporal snapshots of 60%, 20% and 20%, respectively. For the unsupervised LOF, AE, and IForest the initial dataset is split into train and test dataset only,

Table 4
Evaluation metrics used for binary classification.

Measure	Formula	Value Range
Accuracy (or Score)	$\frac{tp+tn}{tp+tn+fp+fn}$ x 100	[0, 100]
Precision	$\frac{tp}{tp+fp}$	[0, 1]
Sensitivity (or Recall)	$\frac{tp}{tp+fn}$	[0, 1]
False Positive Rate (FPR)	$\frac{fp}{fp+tn}$	[0, 1]
F1-score	$\frac{2*(Recall*Precision)}{Recall+Precision}$	[0, 1]

Table 5
Learning model parameters.

Parameters	NN	1-GCN	3-GCN
Neurons in the hidden layer	300	300	300
Activation function	ReLu	ReLu	ReLu
Dropout Value	0.5	0.5	0.5
Optimizer	ADAM	ADAM	ADAM
Learning rate	0.0005	0.0005	0.0005
Batch size	512	-	-
Max Training epochs	100	100	100
Early stopping	10	10	10

with a proportion of 80% and 20%, respectively as they do not require a validation operation. In particular, all attack samples are excluded from the train dataset in order to have the unsupervised models learn normal behaviour only - using an anomaly detection setup called *semi-supervised AD* (Aggarwal, 2017),(Goldstein and Uchida, 2016). For the cluster-based unsupervised models DBSCAN and *k*-NN, the whole dataset is used at once in order to create clusters and detect outliers within them - using an anomaly detection setup called *unsupervised AD* (Aggarwal, 2017),(Goldstein and Uchida, 2016).

For supervised models and semi-supervised AD, each experiment is repeated 5 times with a different composition of their partitions but keeping the described proportions. Hence, for such models, the reported metrics represent an average over 5 repetitions. Standard deviations are reported as well. The process of calculating such metrics over multiple repetitions, however, may introduce unintentional and undesired bias (Forman and Scholz, 2010). There are several methodologies to overcome these issues. As presented in Forman and Scholz (2010), for AUC-ROC, averaging the single values computed in each iteration represents the best solution, whereas for F1-score, it is recommended to apply Equation (5), where TP, FP and FN represent the *total number* of true positives, false positives and false negative over the all repetitions, respectively. For the unsupervised AD setup, results are obtained with a unique execution as the entire dataset is used at once.

In this paper, we exploit the labeled data to compute classical supervised metrics such as Precision, Recall, F1-score and AUC-ROC for unsupervised models too, as in previous studies (Kwon et al., 2019),(Meira et al., 2020),(Perez et al., 2019). In this way, it is easier to compare both supervised and unsupervised results.

$$\frac{2 \times TP}{2 \times TP + FP + FN} \tag{5}$$

The terms $F1\text{-}score_{avg}$ and *AUC-ROC* are used to denote the metrics computed via averaging values obtained in each repetition, whereas $F1\text{-}score_{tp,fp}$ is used to indicate the metric obtained using Equation (5). $F1\text{-}score_{tp,fp}$ does not have a standard deviation as it is calculated combining data from all repetitions.

4.5. Machine learning model architectures and parameters

As explained in Section 3.3, the minimum number of neighbors (N) for allowing SMOTE operations in the SM-hybrid approach is set to 6. In Table 5, configuration parameters for the NN, 1-GCN and 3-GCN learning models are reported. All models are implemented using a similar structure, although they perform the classification task in different ways. Specifically, they are composed of 1 hidden layer of 300 neurons with a Rectified Linear Unit (ReLu) as activation function and 0.50 as dropout value. ADAM optimization algorithms with learning rate 0.0005 are used. For the NN, 512 is chosen as batch size. Generally it is difficult to determine the training duration without generating under-fitting or over-fitting during neural network training. Hence, it is important to fix a parameter called early stopping, which represents the number of consecutive epochs in which the model performance is evaluated (Prechelt, 1998). If performance degrades, the training process is stopped. In our experiments, the maximum number of epochs is set to 100 and the early stopping parameter is set to 10.

For LOF and *k*-NN, the minimum number of neighbors was set to 7. Further, for the *k*-NN, the maximum distance was chosen as 80% of the data distribution, i.e. all the samples that had a distance higher than 80% of the data distribution were considered anomalies. The IForest was implemented with a number of estimators set of 100 and with a threshold anomaly of 0.8. This threshold is used to evaluate the predictions, i.e. if the predicted score of an element exceeds such threshold, the point represents an anomaly, otherwise normal behaviour. For the DBSCAN approach an ϵ (maximum distance between two samples) of 0.2 was chosen. The AE was composed of one hidden layer to reduce the input dimensionality to 45, using a *tanh* function as activation and a *sigmoid* for reconstructing the output. The AE was trained with a batch size of 512 during 50 epochs.

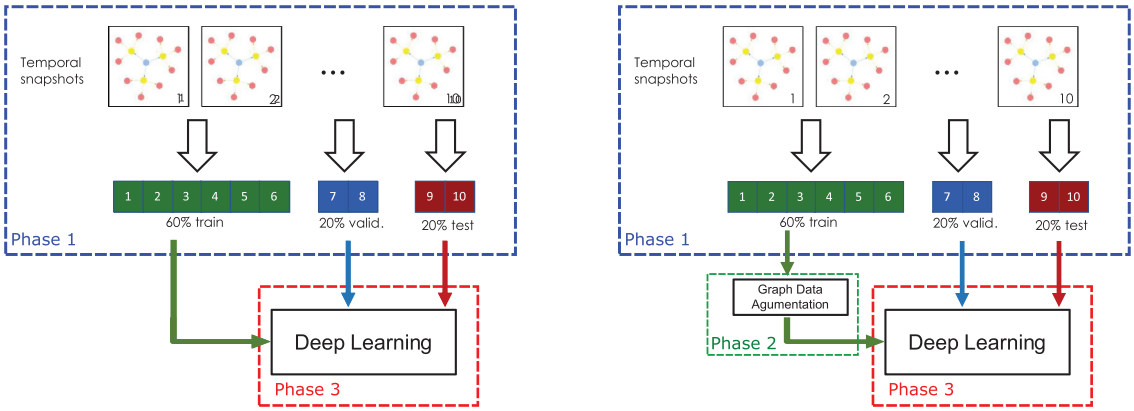
4.6. Experiments

The goal of this study is to classify normal and attacker behaviours in a network traffic dataset using a temporal graph-based representation. For this reason, a novel methodology is introduced applying temporal graph extraction while tackling class imbalance and finally performing classification via Deep Learning-based approaches. Applying this methodology, however, raises the following main questions:

1. *Do the two novel data-level preprocessing approaches correctly address graph class imbalance? How do unsupervised techniques perform with the original, unbalanced dataset?*
2. *How does temporal snapshot size (temporal dissection) affect node behaviour classification?*
3. *Which is the best learning model to be used?*

To address these issues, three main experiments are carried out.

Experiment 1: The aim of this experiment is to compare the performance of 3 graph-based supervised models trained with the imbalanced graph dataset (baseline) versus the same models trained using a balanced dataset that was obtained by applying R-hybrid and SM-hybrid. Furthermore, 5 unsupervised models for anomaly detection are implemented and compared. All the models are trained and tested with temporal TDGs extracted with only one fixed time interval (600s). Both supervised and unsupervised models are first trained and tested with the imbalanced dataset, then (as shown in Figure 7a) for the supervised machine learning models, the two data-level preprocessing approaches are directly applied to the training dataset in order to create a more balanced training population (Phase 2). As described in Section 4.2, the data-level preprocessing operations are performed considering the population of each attack family separately to avoid the creation of



(a) Scheme of the first experiment with an example of 10 temporal snapshots

(b) Scheme of the second experiment with an example of 10 temporal snapshots

Fig. 7. Structure of the experiments.

Table 6
Training population in the first experiment (600*s*).

	Imbalanced Baseline	Balanced R-hybrid	Balanced SM-hybrid
Class	**# samples (%)**	**# samples (%)**	**# samples (%)**
0-class	95.64±0.05	54.72±0.09	54.94±0.07
1-class	4.36±0.05	45.28±0.09	45.06±0.07

small disjuncts. For each temporal snapshot, data-level preprocessing is executed until the sum of attack populations of all families is equal to the initial normal population.

Experiment 2: The goal of this experiment is to evaluate how temporal snapshot sizes affect the definition of node behaviours and their classification. For this reason, three temporal snapshot sizes - 300*s*, 600*s* and 900*s* - are used to split the dataset and extract the temporal TDGs. Then, for each temporal snapshot size, a balanced version of these TDGs is used for training the learning models. The balanced TDGs are obtained by applying the data-level preprocessing operation that shows the best results in *Experiment 1*. Finally, an analysis of how the temporal snapshot size affects the duration of a single training epoch for each model is presented.

Experiment 3: The aim of this experiment is to evaluate which of the three learning approaches NN, 1-GCN and 3-GCN performs best for the node behaviour classification. We compare the best results each model has obtained in previous experiments and also compare their different configurations. Although models are trained to perform a binary classification only (attack/normal behaviour), a study regarding the most detected attack families is carried out in order to highlight benefits and limitations of the presented methodology. For this, the population of a test dataset is evaluated to count how many elements of each attack family are actually classified as attacker node.

5. Experimental Study

In this section, our methodology is validated with a traffic network dataset. In particular, in Section 5.1, Section 5.2 and Section 5.3 the results of the three experiments are detailed, and finally, in Section 5.4, strengths and limitations are discussed.

5.1. Experiment 1: On the goodness of graph data-level preprocessing

Table 6 shows the average training population in 5 repetitions of the first experiment, when a temporal snapshot size of 600*s* was used. Both data-level preprocessing techniques - R-hybrid and SM-hybrid - allowed to address the graph imbalance in the training dataset, converting an initial distribution of 95.64% and 4.36% for the 0-class and 1-class, respectively, to a more balanced distribution of ∼55% and ∼45%. Figure 8 reveals the structure of the training dataset and, in particular the distribution of the attack families that composed the 1-class set, averaged over 5 repetitions. During data-level preprocessing, all attack families were considered separately reshaping their distribution, i.e. the number of samples belonging to each attack family was homogenized. This effect is evident in Figure 8, in which the *Generic, Reconnaissance, Fuzzers* and *DoS* families tended towards a similar representation between 16% and 20% and the *Exploits* family tended to a percentage between 23% and 25%. However, the three other families - *Analysis, Backdoor* and *Shellcode* - did not seem to be affected by data-level preprocessing, and their overall representation was decreased (less than 0.20%). This phenomena may be explained by their graph structures, which may not have satisfied the minimum requirements for being included in the data-level preprocessing, i.e. attacker nodes may not have been involved in subgraphs with at least 60% of attacked nodes.

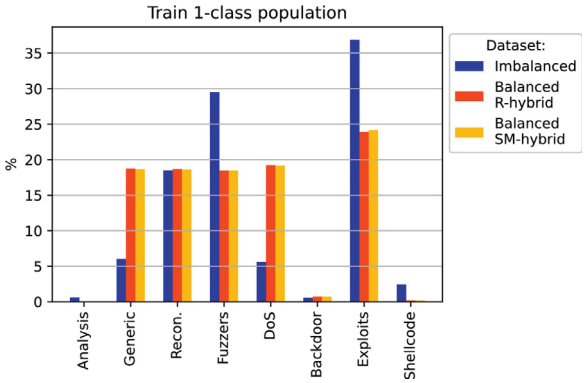


Fig. 8. Attack families distribution in the 1-class training dataset (600*s*).

In Table 7, the classification results obtained after training the learning models with the imbalanced dataset are presented. In particular, these results allow us to define the baseline performance when several supervised as well as unsupervised models are applied. While class imbalance of the training dataset significantly affected the two graph-based models (1-GCN and 3-GCN), it was less relevant for NN and DBSCAN performance, which both reached acceptable values in terms of F1-scores (more than 0.60) and AUC-

Table 7
Comparison between models trained with imbalanced data, using 600*s* as temporal snapshot size (Baseline results).

	Supervised Machine Learning			*Unsupervised Machine Learning*				
	Classification			*Semi-supervised AD*			*Unsupervised AD*	
Metrics	NN	1-GCN	3-GCN	LOF	AE	IForest	DBSCAN	*k*-**NN**
Precision	0.71±0.10	-	0.50±0.20	1.00±0	1.00±0	1.00±0	1.00	1.00
Recall	0.64±0.13	0±0	0.03±0.02	0.28±0.01	0.25±0.01	0.24±0.02	0.48	0.41
F1-score$_{tp,fp}$	0.66	0	0.06	0.44	0.40	0.38	0.64	0.58
F1-score$_{avg}$	0.66±0.06	-	0.06±0.04	0.44±0.01	0.40±0.01	0.38±0.02	0.64	0.58
AUC-ROC	0.81±0.06	0.50±0	0.52±0.01	0.65±0.01	0.63±0.01	0.62±0.01	0.74	0.71

Table 8
Comparison between models trained with balanced (R-hybrid, SM-hybrid) data, using 600*s* as temporal snapshot size.

	Supervised Machine Learning					
	R-hybrid			*SM-hybrid*		
Metrics	**NN**	**1-GCN**	**3-GCN**	**NN**	**1-GCN**	**3-GCN**
Precision	0.53±0.05	0.53±0.03	0.59±0.02	0.54±0.03	0.54±0.03	0.60±0.02
Recall	0.89±0.01	0.77±0.02	0.93±0.01	0.89±0.04	0.81±0.02	0.93±0.02
F1-score$_{tp,fp}$	0.66	0.63	0.73	0.67	0.65	0.73
F1-score$_{avg}$	0.67±0.04	0.63±0.02	0.73±0.01	0.67±0.02	0.65±0.02	0.73±0.02
AUC-ROC	0.93±0.01	0.87±0.01	0.95±0.01	0.93±0.01	0.89±0.01	0.95±0.01

ROC (more than 0.70). More specifically, among the unsupervised methods, the models based on unsupervised AD setup performed better than the ones based on semi-supervised AD. However, all of them (even though reaching best precision scores, i.e. no false positives), showed a high number of false negatives and thus low values of sensitivity (less than 0.50). Among the supervised models, 1-GCN showed very limited performance on the imbalanced data since it learnt to classify all samples as belonging to the 0-class. This effect generated an AUC-ROC value of 0.50, F1-score$_{tp,fp}$ of 0 and made it impossible to calculate the F1-score$_{avg}$. On the other hand, the 3-GCN learnt a few details related to attacker behaviour (AUC-ROC=0.52), which resulted though in rather low F1-scores (F1-score$_{tp,fp}$=F1-score$_{avg}$=0.06). The best results for imbalanced training data were obtained by the NN model with both F1-scores equal to 0.66 and AUC-ROC equal to 0.81.

Table 8 shows the results based on a dataset balanced by using the newly introduced data-level preprocessing techniques. These results demonstrate that a more balanced dataset generated clear improvements when supervised machine learning was used in the classification task. Specifically, the 3-GCN presented the best results in terms of F1-scores and AUC-ROC reaching 0.73 and 0.95, respectively, regardless of the data-level preprocessing used. The 1-GCN improved as well when using the balanced dataset and reached its best values with the SM-hybrid technique (F1-scores=0.65 and AUC-ROC=0.89). The NN model, however, was only slightly affected as its F1-score$_{avg}$ and AUC-ROC improved by only 0.01 and 0.12 points, respectively.

This first experiment indicated that both data-level preprocessing techniques generated classification improvements independently of the supervised machine learning model used. Furthermore, the results obtained in this way were the best results, also when comparing them with the ones obtained from the considered traditional, unsupervised models. More concretely, best results were generated with the SM-hybrid approach. For this reason, only this data-level technique was considered for the second experiment.

5.2. Experiment 2: Comparing different snapshot sizes

Table 9 highlights the effects of the temporal snapshot size on node behaviour classification. The reported metrics were obtained by training and testing the three learning models using distinct temporal snapshot sizes. SM-hybrid is used for balancing the dataset in all cases. All models generated similar values in terms of AUC-ROC regardless of temporal snapshot size, but they diverged with respect to F1-scores. More specifically, all models showed best results using a temporal snapshot size of 600*s*, with an F1-score$_{avg}$ of 0.67, 0.65 and 0.73 for the NN, 1-GCN and 3-GCN, respectively. Increasing the temporal snapshot size to 900*s* generated the worst values in terms of F1-scores and the three models scored 0.06, 0.04 and 0.05 points less compared to their best results, respectively, but kept the AUC-ROC values almost unchanged. On the other hand, decreasing the temporal snapshot size to 300*s* generated slightly lower values in terms of AUC-ROC, where NN, 1-GCN and 3-GCN lost 0.01, 0.02 and 0.01 points, respectively.

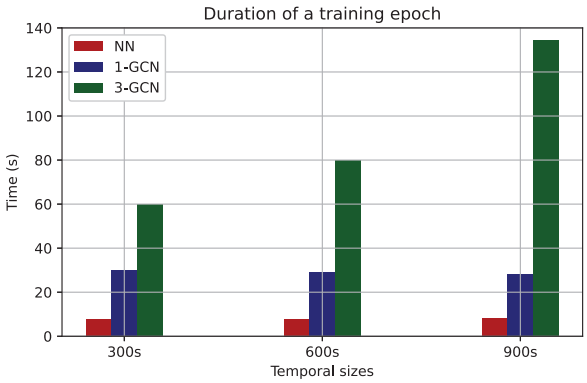


Fig. 9. Duration of a single training epoch when using the SM-hybrid balanced dataset.

Regarding the computational costs of the different approaches when dealing with different temporal snapshot sizes, Figure 9 shows the average duration of an epoch during the training process. Epoch duration during NN and 1-GCN training was similar and did not change visibly when changing temporal snapshot size. However, for the case of 3-GCN training, the duration of an epoch increased almost exponentially with increasing temporal snapshot size, reaching a value of more than two minutes for a single epoch when using a size of 900*s*.

Table 9
Classification metrics obtained in the second experiment by varying the temporal snapshot size and by using SM-hybrid data-level preprocessing.

Data-level SM-hybrid	300s.			600s.			900s.		
Metrics	NN	1-GCN	3-GCN	NN	1-GCN	3-GCN	NN	1-GCN	3-GCN
F1-score$_{tp,fp}$	0.61	0.63	0.70	0.67	0.65	0.73	0.61	0.60	0.67
F1-score$_{avg}$	0.62±0.06	0.63±0.01	0.70±0.02	0.67±0.02	0.65±0.02	0.73±0.02	0.61±0.03	0.61±0.03	0.68±0.03
AUC-ROC	0.92±0.02	0.87±0.02	0.94±0.01	0.93±0.01	0.89±0.01	0.95±0.01	0.92±0	0.89±0.01	0.95±0.01

Table 10
Best results achieved by the three tested learning models.

Metrics	NN	1-GCN	3-GCN
Temporal snapshot size	600s	600s	600s
Data-level	SM-hybrid	SM-hybrid	SM-hybrid
Accuracy	96.12±0.33	96.13±0	96.92±0
Precision	0.54±0.03	0.54±0.03	0.60±0.02
Recall	0.89±0.04	0.81±0.02	0.93±0.02
FPR	0.04±0	0.03±0	0.03±0
F1-score$_{tp,fp}$	0.67	0.65	0.73
F1-score$_{avg}$	0.67±0.02	0.65±0.02	0.73±0.02
AUC-ROC	0.93±0.01	0.89±0.01	0.95±0.01

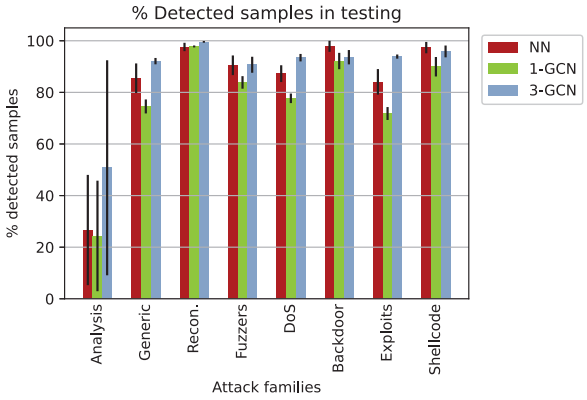


Fig. 10. Attack families as detected by the three learning models in their best configuration.

5.3. Experiment 3: Comparing different learning approaches

In Table 10, the best configuration and the best results of each learning approach are reported. All tested models - NN, 1-GCN, and 3-GCN - showed their best results using the same configuration, i.e. using a temporal snapshot size of 600*s* and using training data balanced with the SM-hybrid technique. Observing this table, the best results over all metrics were obtained using the 3-GCN, which highlights the importance of considering graph relations during node behaviour classification. However, a linear approximation as applied in the 1-GCN did not show benefits since Recall, F1-scores and AUC-ROC metrics were outperformed by the NN model (Table 10).

Figure 10 details the percentages of detected attack families for each of the three learning models when using the best configuration. The 3-GCN detected 7 out of 8 attack families with accuracy higher than 90% and for the *Reconnaissance* family even ∼ 99%. However, 3-GCN had problems detecting samples belonging to the *Analysis* family, as demonstrated by low accuracy values (∼ 50%) and the high variability. The 1-GCN was again outperformed by the NN, which for two families - *Backdoors* and *Shellcode* - also outperformed the 3-GCN classifier. However, NN showed lower accuracy than 3-GCN in detecting *Generic, Fuzzers, DoS* and *Exploits* families (between 85% to 90%), and a clear deficit regarding detection of the *Analysis* family (∼ 27%)

5.4. Discussion

In this paper, we aimed to classify node behaviours in network traffic data by transforming the given time series data into graphs. As the obtained graph-based dataset is characterized by strong class imbalance, we tested two different approaches, one based on supervised ML and the other based on unsupervised ML (anomaly detection, AD). Both approaches initially showed low recall scores and thus lower overall performance (F1-scores and AUC-ROC). For this reason, we investigated the effects of class imbalance in the input dataset and presented two new methods to tackle this issue. Further, we analysed how the temporal snapshot size used for extracting the graphs affects the classification results.

Regarding class imbalance, it was found that the NN implementation was generally not strongly affected and aligned with the unsupervised AD models, while both graph-based learning models required the application of data-level preprocessing techniques in order to improve classification performance. As the application of data-level techniques in the graph domain is not straightforward, we propose here two approaches: *R-hybrid* and *SM-hybrid*. Our experiments show that SM-hybrid generally achieved better results. This approach involves RUS, SMOTE and ROS techniques. Results hence demonstrate that class imbalance is indeed an issue when using graph-based approaches and that these data-level techniques need to be carefully set up.

Regarding temporal snapshot size, in our case, interestingly all models performed best with a size of 600*s* and worsened when increasing the size further. This indicates that it is important to carefully choose temporal snapshot sizes when trying to detect attack node behaviour in network data. Furthermore, it should be noted that using graph-based approaches with higher order filter approximation (such as the 3-GCN) with larger temporal snapshot sizes may yield significantly longer training epochs as our results have shown for the example dataset.

In all experiments, the best results with high Accuracy, Recall and AUC-ROC values were obtained with a 3-GCN model, suggesting that exploiting graph relations for classifying node behaviours is a promising approach. However, experiments showed also that these graph learning models need to be applied and configured properly - as a linear approximation (1-GCN) did not seem to be sufficient to take full advantage of the added graph-based information. Interestingly, this model was outperformed (by a few score points) by a NN model that did include graph-based features, but did not consider relations generated by nodes. Furthermore, the NN model proved to be robust against class imbalance and was not affected in terms of training epoch duration by the chosen temporal size.

Interesting performance differences among the three considered models were revealed when looking at the attack families that could be detected. Here, NN seemed to be the most suitable method for detecting *Backdoors* and *Shellcode* attacks, while the 3-GCN approach performed very well for all other attack families apart from *Analysis*. Actually, all compared models had problems in detecting samples belonging to the *Analysis* family. This attack family, as well as the *Backdoor* and *Shellcode* families, are the least represented ones in the training dataset as they do not

participate in the data-level preprocessing. Although our approach translates the attack classification problem into a behavioural classification task, the classification results of these three attack families can be traced back to the class overlap problem that characterizes the UNSW-NB15 dataset. Analyzing the KMeans Intercluster Distance Map of the original dataset (Zoghi, 2020), *Backdoor* and *Shellcode* families suffer from this class overlap problem with *DoS* and *Fuzzers* families, respectively. However, in the case of binary classification, this issue positively affects the results as more represented classes contain information of less represented classes as well. The *Analysis* family, on the other hand, suffers from class overlap with the *Worm* family. This is the class that had been removed due to its low number of samples. Hence, in this case, there are no other attack families that have information regarding the *Analysis* samples, most likely causing these low scores.

While our approach generated overall improvements - especially when graph-based learning was used - the values of F1-scores could potentially be further improved. More precisely, as demonstrated by the third experiment, Precision values are limited generating a loss of classification quality (although FPRs are less than 0.04). This effect could be related to the application of the UNSW-NB15 dataset and its limitations, as introduced in Section 4.2.

5.4.1. Methodology Limitations

Graphs created from network traffic data mainly depend on the shape of the initial dataset, which should contain information about sources and destinations as well as a temporal axis. One advantage of this approach is that the definition of a "node" could be easily adapted according to the problem at hand, promoting its application to other domains. Moreover, the usage of node behaviour helps to maintain a stable supervision of entities, linking behaviors between snapshots. Further, it allows evaluating features that directly depend on the nodes, such as event logs or suspicious actions in the operating system of the device. However, although the presented methodology can be applied to several IT applications, forensic analysis etc., there are specific scenarios in which its application may be limited. These scenarios include, for example, analysing communication data in critical infrastructures (energy, industry, etc.) where timing is fundamental in order to apply countermeasures and reduce the impact of a cyber-attack very quickly. In these cases, operators typically require reaction times shorter than the temporal snapshots used here for extracting node behaviours. Hence, in these scenarios, traditional intrusion detection systems (IDSs) based on time-series data may be more effective (see next section), which, however, do not provide information regarding node behaviours.

There are also other limitations in our approach that should be discussed. On the one hand, the introduced data-level preprocessing operations are designed for balancing graph datasets in which the information is fragmented and hence we focus on increasing the substructures in which the minority class is present. This is the case of temporal TDGs and network traffic data, but adaptations will likely be required when the problem presents a unique graph where all nodes are connected among them. On the other hand, in terms of the GCN implementation, it is clear from the results that a higher order convolutional filter improves the ability of the classifier, but at the same time it substantially increases the training time.

5.4.2. UNSW-NB15 Comparative Study

This work introduces a novel methodology to analyze a network traffic dataset using a graph-based approach for the classification of node behaviour. The idea is not to directly classify network traffic itself (punctual or specific information), but to evaluate *who* has generated such flows by classifying node behaviour in a temporal snapshot (summary information). With this graph-based representation, it is therefore not possible to directly apply temporal deep learning technologies like 1D-CNN, Long Short-Term Memories (LSTMs) or Recurrent Neural Networks (RNNs) as the information that should be classified is not represented by a point in a time series but by a complex graph composed of distinct nodes that change their individual behaviour (state) over time.

When comparing our work with others in the literature, we did not find any study directly related to our work. To the best of our knowledge, our approach has not been explored before. For this reason, Table 11 reports previous works that use the UNSW-NB15 dataset for implementing different intrusion detection systems (IDSs). In these studies the network traffic dataset is analyzed as a *time series* with the aim of instantly detecting malicious flows (typical IDS task). Hence, a direct one-to-one comparison with our *graph-based* approach analysing node behaviours is difficult and potentially unfair. In particular, previous implementations are mainly based on ML and Deep Learning (DL) models such as Support Vector Machines (SVM), Random Forest (RF), Naive Bayes (NB), Multilayer perceptron (MLP), CNNs, 1D-CNN and LSTMs.

As shown in Table 11, the accuracy obtained with the proposed node behaviour approach is quite similar to the one obtained by directly classifying network connections; in fact, in their best implementation they reach ∼96% and ∼99%, respectively. However, analyzing the results in terms of Precision, Recall and F1 score, it is clear that the node behaviour classification is a more complex and challenging task than the one based on network connections. In particular, in its best configuration, the introduced node behaviour approach reaches values of 0.60, 0.93 and 0.73, for Precision, Recall and F1, respectively, while network connection classifiers show values higher than 0.90 in all the three metrics. Yet, it is to be noted that the network traffic approach generates very variable results depending on the approximation and the algorithm used. In terms of training time, some ML approaches were the fastest with a duration of a couple of seconds only (NB), while more complex approaches like CNN required more than 1 hour. The 3-GCN implemented in this work required more than 2 hours for training the model, while the 1-GCN was faster than a simple CNN (Jiang et al., 2020) and was totally aligned with the training time of a CNN combined with a BiLSTM (Jiang et al., 2020). Note that he reported values (strongly) depend on the computational resources used, which varies in the different works.

Finally, we would like to point out again that a one-to-one comparison between our graph-based approach and previously published models is challenging as these models predominantly analyse time-series data of communication flows between actors in the network to detect suspicious individual flows. Additionally, although all the analyzed approaches start from the same initial dataset, the extraction of TDGs for the node behaviour classification alters the magnitude of the imbalance problem, increasing the difficulty to compare the results. Furthermore, the models based on time series do not provide broader information regarding the actors, i.e., regarding the nodes themselves and their behaviour over time. Unlike our graph-based approach they cannot classify an entire node's behaviour as a potentially malicious actor in the network. Their results are thus instantaneous and therefore quicker but without providing the bigger picture concerning all nodes' behaviours. With our GCN-based models, one could identify malicious actors within a network and then make decisions regarding the isolation or exclusion of this actor in order to restore the cybersecurity of the entire network. In fact, future elaborate network monitoring systems may provide a combination of both detection capabilities: time-series-based IDS for quick, short-term malicious flow detection and longer term graph-based node analysis to pinpoint potentially malicious actors in the network based on their behaviour.

Table 11
Comparison of the GCNs implemented in this study with existing IDS models that analyse the UNSW-NB15 dataset (note, however, that results between node behavior classification and network connection classification are not directly comparable).

Scope/ Classification	Data Type	Model	Combined	Ref.	Accuracy (%)	Precision	Recall	F1	AUC	Training time (s)
Node behaviour	Graph	1-GCN	-	-	96.13	0.54	0.81	0.65	0.89	2,863
		3-GCN	-	-	96.92	0.60	0.93	0.73	0.95	8,081
Network connection	Time series	SVM	-	Belouch et al. (2018)	92.28	-	0.92	-	-	38.91
		Random	-	Belouch et al. (2018)	97.49	-	0.94	-	-	5.69
		Forest	-	Zhou et al. (2020)	-	0.78	0.99	0.87	0.82	-
			-	Segurola-Gil et al. (2021)	99.00	0.94	0.99	0.97	-	217.97
			-	Azizjon et al. (2020)	87.82	0.90	0.87	0.87	-	-
			+SGM	Zhang et al. (2020)	98.68	0.91	-	0.95	-	9.28
		Decision	-	Belouch et al. (2018)	95.82	-	0.93	-	-	4.80
		Tree	+SVD	Ugwu et al. (2021)	92.08	0.76	0.81	0.79	-	-
		Naive	-	Belouch et al. (2018)	74.19	-	0.92	-	-	2.25
		Bayes	-	Zhou et al. (2020)	-	0.75	0.98	0.85	0.81	-
			+AE	Segurola-Gil et al. (2021)	98.00	0.91	0.98	0.94	-	169.15
			+SVD	Ugwu et al. (2021)	86.31	0.78	0.70	0.73	-	-
		MLP	-	Segurola-Gil et al. (2021)	99.00	0.92	0.99	0.95	-	282.89
			+SGM	Zhang et al. (2020)	98.74	0.91	-	0.95	-	7.90
		1D-CNN	-	Azizjon et al. (2020)	91.20	0.88	0.96	0.92	-	-
			+LSTM	Azizjon et al. (2020)	89.93	0.86	0.95	0.90	-	-
		CNN	-	Dhillon and Haque (2021)	92.16	-	-	-	-	-
			-	Jiang et al. (2020)	74.65	0.80	0.76	0.78	-	4,522.56
			+SGM	Zhang et al. (2020)	98.82	0.92	-	0.96	-	48.56
			+LSTM	Dhillon and Haque (2021)	98.30	-	-	-	-	-
			+LSTM	Zhou et al. (2020)	-	0.80	0.96	0.87	0.83	-
			+BiLSTM	Jiang et al. (2020)	76.82	0.82	0.79	0.80	-	2,750.47
		LSTM	-	Ugwu et al. (2021)	84.68	0.61	0.40	0.48	-	-
			-	Zhou et al. (2020)	-	0.81	0.99	0.89	0.85	-
			+SVD	Ugwu et al. (2021)	94.28	0.86	0.80	0.84	-	-
			Variational	Zhou et al. (2020)	-	0.86	0.98	0.91	0.90	-

6. Conclusions and Future work

In this study, we present a novel methodology that converts an attack classification problem into a node behaviour classification problem, thereby highlighting the importance of understanding and correctly manipulating the input data and properly configuring the classification model. Our approach allows extracting temporal graph-based information (temporal TDGs) from network traffic data while focusing on micro-dynamics and the evolution of node behaviours. Two novel techniques for addressing class imbalance in the graph domain were proposed (R-hybrid and SM-hybrid) and proved to be suitable to reduce class imbalance while minimizing changes in the graph topology. When temporally dissecting the given network traffic data to unveil network micro-dynamics, we investigated the effect of the temporal snapshot size on the classification of node behaviour. Finally, three different Deep Learning approaches - Neural Network, 1-GCN, and 3-GCN - were implemented and compared with more traditional unsupervised anomaly detection methods that do not require a balanced dataset. Overall, the methods presented in this paper showed promising results that can be summarized as follows:

- It is possible to convert time series information into graph-based structures by properly defining the concept of node/entity and link/edge. Then, edge information can be combined to characterize each node behaviour;
- The unsupervised models for anomaly detection trained as semi-supervised AD together with the 1-GCN and 3-GCN were most affected by class imbalance;
- The NN together with the DBSCAN (unsupervised AD) showed good results for the imbalanced data;
- Novel proposed data-level preprocessing techniques - R-hybrid and SM-hybrid - successfully solved the class imbalance problem in graph data and yielded good classification results. SM-hybrid generated the best results.
- Temporal snapshot size is relevant when analysing network traffic data using a graph-based approach. All learning models generated the best classification results using a temporal snapshot size of 600*s* for the given dataset.
- Increasing the temporal snapshot size heavily affected single training epoch duration for the 3-GCN model (exponential upward trend), while NN and 1-GCN training epoch duration was less affected;
- The 3-GCN model including graph-based features showed best overall classification performance. However, the graph-based model applying linear approximation (1-GCN) was outperformed by the NN approach, which did not use any graph relations, regardless of the temporal snapshot size;
- Among the 8 considered attack families, the 3-GCN was able to detect 7 of them with an Accuracy above 90%. However, the attack families *Backdoors* and *Shellcode* could best be detected via NN. None of the learning models were able to detect *Analysis* attacks well.

The presented graph data-level preprocessing approaches were developed in order not to modify the graph structures and topology. Further, they needed to be applicable in situations in which graph information is fragmented and the minority class creates highly-represented substructures. As future work, it may be interesting to modify these techniques so that they can be applied in cases where entities are fairly mixed with other classes (as it is the case for the *Analysis, Backdoor* and *Shellcode* families in our experiment). Furthermore, the precision of the classification could be improved by applying clustering operations after the TDGs creation in order to aggregate similar behaviours, reducing noisy data and enhancing small interactions between the nodes. The effectiveness of distinct clustering algorithms should be compared in order to understand how they affect the initial population as well as the final classification. Finally, an approach based on Generative Adversarial Networks (GANs) for creating new synthetic node behaviour

and their connections within the graph could be a solution for improving the variety of attack class samples.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

F. Zola: Conceptualization, Investigation, Methodology, Validation, Visualization, Writing – original draft. **L. Segurola-Gil:** Investigation, Data curation. **J.L. Bruse:** Visualization, Writing – review & editing. **M. Galar:** Visualization, Writing – review & editing. **R. Orduna-Urrutia:** Conceptualization, Supervision.

References

Aggarwal, C.C., 2017. An introduction to outlier analysis. In: Outlier analysis. Springer, pp. 1–34.

Agrafiotis, I., Nurse, J.R.C., Goldsmith, M., Creese, S., Upton, D., 2018. A taxonomy of cyber-harms: Defining the impacts of cyber-attacks and understanding how they propagate. Journal of Cybersecurity 4 (1), tyy006.

Akoglu, L., Tong, H., Koutra, D., 2015. Graph based anomaly detection and description: a survey. Data mining and knowledge discovery 29 (3), 626–688.

Azizjon, M., Jumabek, A., Kim, W., 2020. 1d CNN based network intrusion detection with normalization on imbalanced data. In: 2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC). IEEE, pp. 218–224.

Belouch, M., El Hadaj, S., Idhammad, M., 2018. Performance evaluation of intrusion detection based on machine learning using apache spark. Procedia Computer Science 127, 1–6.

Bollobás, B., 2013. Modern graph theory, volume 184. Springer Science & Business Media.

Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J., 2000. Lof: identifying density-based local outliers. In: Proceedings of the 2000 ACM SIGMOD international conference on Management of data, pp. 93–104.

Chandola, V., Banerjee, A., Kumar, V., 2009. Anomaly detection: A survey. ACM computing surveys (CSUR) 41 (3), 1–58.

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

Coley, C.W., Jin, W., Rogers, L., Jamison, T.F., Jaakkola, T.S., Green, W.H., Barzilay, R., Jensen, K.F., 2019. A graph-convolutional neural network model for the prediction of chemical reactivity. Chemical science 10 (2), 370–377.

Crovella, M., Kolaczyk, E., 2003. Graph wavelets for spatial traffic analysis. In: IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428), volume 3. IEEE, pp. 1848–1857.

Defferrard, M., Bresson, X., Vandergheynst, P., 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in neural information processing systems, pp. 3844–3852.

Dhillon, H., Haque, A., 2021. Towards network traffic monitoring using deep transfer learning. arXiv e-prints. arXiv:2101.00731.

Djidjev, H., Sandine, G., Storlie, C., Vander Wiel, S., 2011. Graph based statistical analysis of network traffic. In: Proceedings of the Ninth Workshop on Mining and Learning with Graphs.

Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al., 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In: kdd, volume 96, pp. 226–231.

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

Fernández, A., García, S., Galar, M., Prati, R.C., Krawczyk, B., Herrera, F., 2018. Learning from imbalanced data sets, volume 11. Springer.

Forman, G., Scholz, M., 2010. Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement. Acm Sigkdd Explorations Newsletter 12 (1), 49–57.

Formosa, P., Wilson, M., Richards, D., 2021. A principlist framework for cybersecurity ethics. Computers & Security 102382.

Gao, H., Cheng, S., Zhang, W., 2021. Gdroid: Android malware detection and classification with graph convolutional network. Computers & Security 106, 102264.

Garcia-Garcia, A., Orts-Escolano, S., Oprea, S., Villena-Martinez, V., Martinez-Gonzalez, P., Garcia-Rodriguez, J., 2018. A survey on deep learning techniques for image and video semantic segmentation. Applied Soft Computing 70, 41–65.

Goldstein, M., Uchida, S., 2016. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. PloS one 11 (4), e0152173.

Hamilton, W., Ying, Z., Leskovec, J., 2017. Inductive representation learning on large graphs. In: Advances in neural information processing systems, pp. 1024–1034.

Hart, P., 1968. The condensed nearest neighbor rule (corresp.). IEEE transactions on information theory 14 (3), 515–516.

He, H., Bai, Y., Garcia, E.A., Li, S., 2008. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In: 2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence). IEEE, pp. 1322–1328.

Hu, P., Lau, W.C.. A survey and taxonomy of graph sampling.

Iliofotou, M., Faloutsos, M., Mitzenmacher, M., 2009a. Exploiting dynamicity in graph-based traffic analysis: Techniques and applications. In: Proceedings of the 5th international conference on Emerging networking experiments and technologies, pp. 241–252.

Iliofotou, M., Kim, H.c., Faloutsos, M., Mitzenmacher, M., Pappu, P., Varghese, G., 2009b. Graph-based p2p traffic classification at the internet backbone. In: IEEE INFOCOM Workshops 2009. IEEE, pp. 1–6.

Iliofotou, M., Pappu, P., Faloutsos, M., Mitzenmacher, M., Singh, S., Varghese, G., 2007. Network monitoring using traffic dispersion graphs (tdgs). In: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, pp. 315–320.

Japkowicz, N., Stephen, S., 2002. The class imbalance problem: A systematic study. Intelligent data analysis 6 (5), 429–449.

Jiang, J., Chen, J., Gu, T., Choo, K.K.R., Liu, C., Yu, M., Huang, W., Mohapatra, P., 2019. Anomaly detection with graph convolutional networks for insider threat and fraud detection. In: MILCOM 2019-2019 IEEE Military Communications Conference (MILCOM). IEEE, pp. 109–114.

Jiang, K., Wang, W., Wang, A., Wu, H., 2020. Network intrusion detection combined hybrid sampling with deep hierarchical network. IEEE Access 8, 32464–32476.

Jin, B., Gao, C., He, X., Jin, D., Li, Y., 2020. Multi-behavior recommendation with graph convolutional networks. In: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 659–668.

Jin, Y., Sharafuddin, E., Zhang, Z.L., 2009. Unveiling core network-wide communication patterns through application traffic activity graph decomposition. ACM SIGMETRICS Performance Evaluation Review 37 (1), 49–60.

Kermarrec, A.M., Le Merrer, E., Sericola, B., Trédan, G., 2011. Second order centrality: Distributed assessment of nodes criticity in complex networks. Computer Communications 34 (5), 619–628.

Khraisat, A., Gondal, I., Vamplew, P., Kamruzzaman, J., 2019. Survey of intrusion detection systems: techniques, datasets and challenges. Cybersecurity 2 (1), 20.

Kipf, T.N., Welling, M., 2017. Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (ICLR).

Kwon, D., Kim, H., Kim, J., Suh, S.C., Kim, I., Kim, K.J., 2019. A survey of deep learning-based network anomaly detection. Cluster Computing 22 (1), 949–961.

Leevy, J.L., Khoshgoftaar, T.M., Bauder, R.A., Seliya, N., 2018. A survey on addressing high-class imbalance in big data. Journal of Big Data 5 (1), 1–30.

Leskovec, J., Faloutsos, C., 2006. Sampling from large graphs. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 631–636.

Leung, K., Leckie, C., 2005. Unsupervised anomaly detection in network intrusion detection using clusters. In: Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38, pp. 333–342.

Li, J., Izakian, H., Pedrycz, W., Jamal, I., 2021. Clustering-based anomaly detection in multivariate time series data. Applied Soft Computing 100, 106919.

Liao, Y., Vemuri, V.R., 2002. Use of k-nearest neighbor classifier for intrusion detection. Computers & security 21 (5), 439–448.

Liu, F.T., Ting, K.M., Zhou, Z.H., 2012. Isolation-based anomaly detection. ACM Transactions on Knowledge Discovery from Data (TKDD) 6 (1), 1–39.

Long, Y., Wu, M., Kwoh, C.K., Luo, J., Li, X., 2020. Predicting human microbe-drug associations via graph convolutional network with conditional random field. Bioinformatics.

Meira, J., Andrade, R., Praça, I., Carneiro, J.a., Bolón-Canedo, V., Alonso-Betanzos, A., Marreiros, G., 2020. Performance evaluation of unsupervised techniques in cyber-attack anomaly detection. Journal of Ambient Intelligence and Humanized Computing 11 (11), 4477–4489.

Monshizadeh, M., Khatri, V., Atli, B.G., Kantola, R., Yan, Z., 2019. Performance evaluation of a combined anomaly detection platform. IEEE Access 7, 100964–100978.

Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., Bronstein, M.M., 2017. Geometric deep learning on graphs and manifolds using mixture model cnns. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5115–5124.

Moustafa, N., 2017. Designing an online and reliable statistical anomaly detection framework for dealing with large high-speed network traffic. University of New South Wales, Canberra, Australia.

Moustafa, N., Slay, J., 2015. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In: 2015 military communications and information systems conference (MilCIS). IEEE, pp. 1–6.

Moustafa, N., Slay, J.. Rcnf: Real-time collaborative network forensic scheme for evidence analysis.

Nagaraja, S., Mittal, P., Hong, C.Y., Caesar, M., Borisov, N., 2010. Botgrep: Finding p2p bots with structured graph analysis. In: USENIX security symposium, volume 10, pp. 95–110.

Nguyen, H.M., Cooper, E.W., Kamei, K., 2011. Borderline over-sampling for imbalanced data classification. International Journal of Knowledge Engineering and Soft Data Paradigms 3 (1), 4–21.

Oba, T., Taniguchi, T.. Graph convolutional network-based suspicious communication pair estimation for industrial control systems.

Omar, S., Ngadi, A., Jebur, H.H., 2013. Machine learning techniques for anomaly detection: an overview. International Journal of Computer Applications 79 (2).

Pei, X., Yu, L., Tian, S., 2020. Amalnet: A deep learning framework based on graph convolutional networks for malware detection. Computers & Security 93, 101792.
Perez, D., Alonso, S., Moran, A., Prada, M.A., Fuertes, J.J., Dominguez, M., 2019. Comparison of network intrusion detection performance using feature representation. In: International Conference on Engineering Applications of Neural Networks. Springer, pp. 463–475.
Prechelt, L., 1998. Early stopping-but when? In: Neural Networks: Tricks of the trade. Springer, pp. 55–69.
Rong, Y., Huang, W., Xu, T., Huang, J.. Dropedge: Towards deep graph convolutional networks on node classification.
Sarker, I.H., Kayes, A., Badsha, S., Alqahtani, H., Watters, P., Ng, A., 2020. Cybersecurity data science: an overview from machine learning perspective. Journal of Big Data 7 (1), 1–29.
Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G., 2008. The graph neural network model. IEEE Transactions on Neural Networks 20 (1), 61–80.
Segurola-Gil, L., Zola, F., Echeberria-Barrio, X., Orduna-Urrutia, R.. Nbcoded: network attack classifiers based on encoder and naive bayes model for resource limited devices.
Silva, T.C., Zhao, L., 2016. Machine learning in complex networks, volume 1. Springer.
Stivala, A.D., Koskinen, J.H., Rolls, D.A., Wang, P., Robins, G.L., 2016. Snowball sampling for estimating exponential random graph models for large networks. Social Networks 47, 167–188.
Stumpf, M., Wiuf, C., May, R., 2005. Subnets of scale-free networks are not scale-free: Sampling properties of networks. Proceedings of the National Academy of Sciences of the United States of America 102, 4221–4224. doi:10.1073/pnas.0501179102.
Sun, X., Wang, Z., Yang, J., Liu, X., 2020a. Deepdom: Malicious domain detection with scalable and heterogeneous graph convolutional networks. Computers & Security 99, 102057.
Sun, X., Yang, J., Wang, Z., Liu, H., 2020b. Hgdom: Heterogeneous graph convolutional networks for malicious domain detection. In: NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium. IEEE, pp. 1–9.
Tan, C.L., Chiew, K.L., Yong, K.S.C., Abdullah, J., Sebastian, Y., et al., 2020. A graph-theoretic approach for the detection of phishing webpages. Computers & Security 95, 101793.
Tomek I., et al. Two modifications of CNN1976;.
Ugwu, C.C., Obe, O.O., Popoola, O.S., Adetunmbi, A.O., 2021. A distributed denial of service attack detection system using long short term memory with singular value decomposition. In: 2020 IEEE 2nd International Conference on Cyberspac (CYBER NIGERIA). IEEE, pp. 112–118.
Van Schaik, P., Renaud, K., Wilson, C., Jansen, J., Onibokun, J., 2020. Risk as affect: The affect heuristic in cybersecurity. Computers & Security 90, 101651.
Wang, W., Shang, Y., He, Y., Li, Y., Liu, J., 2020. Botmark: Automated botnet detection with hybrid analysis of flow-based and graph-based traffic behaviors. Information Sciences 511, 284–296.
Wehmuth, K., Ziviani, A., Fleury, E., 2015. A unifying model for representing time–varying graphs. In: 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA). IEEE, pp. 1–10.
Weiss, G.M., 2010. The impact of small disjuncts on classifier learning. In: Data Mining. Springer, pp. 193–226.
Wu, L., Sun, P., Hong, R., Fu, Y., Wang, X., Wang, M.. SocialGCN: An efficient graph convolutional network based model for social recommendation.
Wu, Y., Cao, N., Archambault, D., Shen, Q., Qu, H., Cui, W., 2016. Evaluation of graph sampling: A visualization perspective. IEEE transactions on visualization and computer graphics 23 (1), 401–410.
Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y., 2020. A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems.
Xie, T., Grossman, J.C., 2018. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. Physical review letters 120 (14), 145301.
Xin, Y., Kong, L., Liu, Z., Chen, Y., Li, Y., Zhu, H., Gao, M., Hou, H., Wang, C., 2018. Machine learning and deep learning methods for cybersecurity. IEEE Access 6, 35365–35381.
Yao, Y., Su, L., Lu, Z., 2018. DeepGFL: Deep feature learning via graph for attack detection on flow-based network traffic. In: MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM). IEEE, pp. 579–584.
Yen, S.J., Lee, Y.S., 2006. Under-sampling approaches for improving prediction of the minority class in an imbalanced dataset. In: Intelligent Control and Automation. Springer, pp. 731–740.
You, J., Liu, B., Ying, Z., Pande, V., Leskovec, J., 2018. Graph convolutional policy network for goal-directed molecular graph generation. Advances in neural information processing systems.
Zhang, H., Huang, L., Wu, C.Q., Li, Z., 2020. An effective convolutional neural network based on SMOTE and gaussian mixture model for intrusion detection in imbalanced dataset. Computer Networks 177, 107315.
Zhang, H., Wu, C.Q., Gao, S., Wang, Z., Xu, Y., Liu, Y., 2018. An effective deep learning based scheme for network intrusion detection. In: 2018 24th International Conference on Pattern Recognition (ICPR). IEEE, pp. 682–687.
Zhang, S., Tong, H., Xu, J., Maciejewski, R., 2019. Graph convolutional networks: a comprehensive review. Computational Social Networks 6 (1), 1–23.
Zhao, J., Liu, X., Yan, Q., Li, B., Shao, M., Peng, H., 2020a. Multi-attributed heterogeneous graph convolutional network for bot detection. Information Sciences.
Zhao, J., Yan, Q., Liu, X., Li, B., Zuo, G., 2020b. Cyber threat intelligence modeling based on heterogeneous graph convolutional network. In: 23rd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2020), pp. 241–256.
Zhao, T., Liu, Y., Neves, L., Woodford, O., Jiang, M., Shah, N.. Data augmentation for graph neural networks.
Zheng, L., Li, Z., Li, J., Li, Z., Gao, J., 2019. Addgraph: Anomaly detection in dynamic graph using attention-based temporal GCN. In: IJCAI, pp. 4419–4425.
Zhou, C., Paffenroth, R.C., 2017. Anomaly detection with robust deep autoencoders. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, pp. 665–674.
Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M.. Graph neural networks: A review of methods and applications.
Zhou, X., Hu, Y., Liang, W., Ma, J., Jin, Q., 2020. Variational LSTM enhanced anomaly detection for industrial big data. IEEE Transactions on Industrial Informatics 17 (5), 3469–3477.
Zhou, Y., Hu, G., He, W., 2009. Using graph to detect network traffic anomaly. In: 2009 International Conference on Communications, Circuits and Systems. IEEE, pp. 341–345.
Zoghi, Z., 2020. Ensemble Classifier Design and Performance Evaluation for Intrusion Detection Using UNSW-NB15 Dataset. University of Toledo.

Francesco Zola obtained his bachelor's degree in Telecommunication Engineering at University of Cassino and Southern Lazio, Italy, in 2012 and his master's degree in Computer Science in the same university, in 2015. He works as associate researcher and data scientist in Vicomtech in the department of Digital Security applying machine learning in cybersecurity projects. In 2019 he started his doctorate in collaboration with the Public University of Navarre (UPNA), focusing his works on graph analysis, machine learning, data augmentation and classification. Francesco is involved in several European and industrial projects related to cybersecurity, blockchain analysis and anomaly detection.

Lander Segurola-Gil is a mathematician who obtained his bachelor's degree in the University of País Vasco - Euskal Herriko Unibertsitatea (UPV-EHU) and received a master's degree in Mathematics and Applications from the University of Madrid (UAM). Since 2019, he works as a research assistant in Vicomtech in the department of Digital Security. His research interests are Naive Bayes networks, attack detection and machine learning applied to time series.

Dr. Jan L. Bruse graduated in Mechanical Engineering at RWTH Aachen University, Germany, in 2013 and received his doctorate in Biomedical Engineering from University College London, United Kingdom, in 2017. His PhD thesis combines Medical Image Analysis, Computational Simulation and Machine Learning for the development of Clinical Decision Support Systems. Jan has participated in several international and interdisciplinary research collaborations, has multiple peer-reviewed publications in journals and congresses indexed in the engineering and clinical sector and is reviewer for several international journals. Since September 2017, Jan is Research Engineer at Vicomtech, Spain, in the area of Data Intelligence for Energy and Industrial Processes where he develops Artificial Intelligence (AI) and Machine Learning platforms within the context of Industry 4.0 projects.

Dr. Mikel Galar received the MSc and PhD degrees in Computer Science in 2009 and 2012, both from the Public University of Navarre (UPNA), Spain. He is currently an associate professor at the Department of Statistics, Computer Science and Mathematics at the UPNA. He is the author of 35 published original articles in international journals and more than 50 contributions to conferences. He is also reviewer of more than 35 international journals. His research interests are machine learning, data mining, classification, fuzzy systems and big data. He is a member of the IEEE, the European Society for Fuzzy Logic and Technology (EUSFLAT) and the Spanish Association of Artificial Intelligence (AEPIA). He has received the extraordinary prize for his PhD thesis from the Public University of Navarre and the 2013 IEEE Transactions on Fuzzy System Outstanding Paper Award for the paper “A New Approach to Interval-Valued Choquet Integrals and the Problem of Ordering in Interval-Valued Fuzzy Set Applications” (bestowed in 2016)

Dr. Raul Orduna-Urrutia received his degree in Computer Engineering at the Faculty of Informatics of San Sebastian by the University of the Basque Country (UPV/EHU) and obtained a PhD degree in Computer Science and Artificial Intelligence at the School of Industrial and Telecommunications Engineering (ETSIIT) of the Public University of Navarre (UPNA). He is currently an associate professor at the Department of Statistics and Computer Science at the UPNA, and, at the same time, the Digital Security Director in Vicomtech. He has taken part or led projects related with ethical hacking, forensic analysis, malware analysis, access control and cryptography. He has participated in more than 6 successful funded projects, is the author of 6 published original articles in international journals. He is also a reviewer of Fuzzy Sets and Systems.

3. *Attacking Bitcoin anonymity: Generative Adversarial Networks for improving Bitcoin entity classification*

The work related to this part is in:

- Zola F., Segurola Gil L., Bruse J.L., Galar M. and Orduna Urrutia R., *Attacking Bitcoin anonymity: Generative Adversarial Networks for improving Bitcoin entity classification*, Appl Intell (2022), DOI: 10.1007/s10489-022-03378-7

- Status: Published.
- Impact Factor (JCR 2020): 5.086.
- Knowledge area:
  - Computer Science, Artificial Intelligence. Ranking 35/139 (Q2).

Applied Intelligence
https://doi.org/10.1007/s10489-022-03378-7

Attacking Bitcoin anonymity: generative adversarial networks for improving Bitcoin entity classification

Francesco Zola[1,2] · Lander Segurola-Gil[1] · Jan L. Bruse[1] · Mikel Galar[2] · Raul Orduna-Urrutia[1]

Accepted: 11 February 2022

Abstract

Classification of Bitcoin entities is an important task to help Law Enforcement Agencies reduce anonymity in the Bitcoin blockchain network and to detect classes more tied to illegal activities. However, this task is strongly conditioned by a severe class imbalance in Bitcoin datasets. Existing approaches for addressing the class imbalance problem can be improved considering generative adversarial networks (GANs) that can boost data diversity. However, GANs are mainly applied in computer vision and natural language processing tasks, but not in Bitcoin entity behaviour classification where they may be useful for learning and generating synthetic behaviours. Therefore, in this work, we present a novel approach to address the class imbalance in Bitcoin entity classification by applying GANs. In particular, three GAN architectures were implemented and compared in order to find the most suitable architecture for generating Bitcoin entity behaviours. More specifically, GANs were used to address the Bitcoin imbalance problem by generating synthetic data of the less represented classes before training the final entity classifier. The results were used to evaluate the capabilities of the different GAN architectures in terms of training time, performance, repeatability, and computational costs. Finally, the results achieved by the proposed GAN-based resampling were compared with those obtained using five well-known data-level preprocessing techniques. Models trained with data resampled with our GAN-based approach achieved the highest accuracy improvements and were among the best in terms of precision, recall and f1-score. Together with Random Oversampling (ROS), GANs proved to be strong contenders in addressing Bitcoin class imbalance and consequently in reducing Bitcoin entity anonymity (overall and per-class classification performance). To the best of our knowledge, this is the first work to explore the advantages and limitations of GANs in generating specific Bitcoin data and "attacking" Bitcoin anonymity. The proposed methods ultimately demonstrate that in Bitcoin applications, GANs are indeed able to learn the data distribution and generate new samples starting from a very limited class representation, which leads to better detection of classes related to illegal activities.

Keywords Bitcoin address classification · Entity anonymity attack · Entity classification · Generative adversarial networks (GAN) · Class imbalance problem

1 Introduction

Bitcoin represents a pseudo-anonymous peer-to-peer network, which allows its users to communicate through transactions stored in a public ledger called blockchain [1]. To date, Bitcoin is the most frequently used cryptocurrency for concealing illicit activities as quantified in traffic of around $76 billion per year [2]. Users typically feel shielded by Bitcoin anonymity and at the same time conceive it as a convenient payment mechanism [3]. Furthermore, although the volatility of cryptomarkets introduces high risks, entrepreneurs and corporate executives have high expectations regarding blockchain technology. Several recommendations for blockchain platforms can be found in [4]. An escalation of illegal activities and the goal of improving the network's resilience to cyber-attacks have led researchers and Law Enforcement Agencies (LEAs) to investigate how to reduce anonymity within the Bitcoin blockchain network [5]. Anonymity can be decreased through Bitcoin entity classification [6, 7], which aims to detect and classify entities' behavioural patterns within the network. However, this classification problem - typically addressed via supervised machine learning approaches - strongly depends on the initial labelled Bitcoin dataset,

✉ Francesco Zola
fzola@vicomtech.org

Extended author information available on the last page of the article.

Published online: 01 April 2022

which allows the definition of singular entity behaviour. These kinds of datasets are often characterized by a non-homogeneously distributed population, which means that some entity classes present in the dataset are more populated than others, causing a severe *class imbalance problem*. Especially for supervised machine learning problems, this phenomenon dramatically affects the quality of the learning system. Classifiers trained using imbalanced datasets are usually biased in favour of the majority classes and fail to detect underrepresented ones [8].

The class imbalance problem becomes even more relevant in applications where it is hard to detect and collect new observations as is typically the case for Bitcoin-related data. The lack of data, usually related to samples potentially associated with illicit activities, negatively affects the performance of a classifier for a behavioural prediction system [9].

Traditional approaches used to address dataset imbalance problems can be clustered into four groups [8]: cost-sensitive learning, data-level preprocessing, algorithm-level approaches, and ensemble learning. Cost-sensitive learning uses cost values for penalizing misclassifications of some classes to improve their importance during the training phase [10]. Data-level approaches are based on adjusting the initial distribution to construct a balanced training dataset (resampling). Algorithm-level techniques try to modify existing algorithms to address the class imbalance problem directly inside the learning algorithm itself [11]. Ensemble learning is composed of techniques that train different classifiers with the original data and then combine their results for the final classification [8].

Cost-sensitive and data-level approaches, as well as a combination of both methods, are the most used techniques when deep learning models are trained [12]. In particular, cost-sensitive modifications of the back-propagation learning are made in order to improve the sensitivity of the minority class. This operation promotes the classification of samples in the minority class over the majority ones, as shown in several works [12, 13].

Other interesting and more complex approaches for addressing the imbalance problem are based on the implementation of generative models for enhancing the size and quality of the training data [14]. These approaches have been widely and very successfully used for image [15] and video [16] generation, since they are able to learn underlying true data distributions from limited available samples. In fact, as presented in [17], adversarial technology, and more specifically generative models, allow access and unlock hidden information in a dataset. Moreover, these models are typically used for enhancing the fairness in the original dataset to avoid bias in the classification [18].

For this reason, in this paper, we introduce and analyse a method based on adversarial learning (generative model) to tackle the Bitcoin class imbalance problem, with the ultimate goal of improving the performance of the final classifier. Motivated by their good results in other domains and their promising capabilities to learn complex data distributions, we use here Generative Adversarial Networks (GANs) [19] to create new synthetic samples balancing the class distributions in the original dataset, and then use the balanced dataset to train the classifier. In this way, we can evaluate how the additional synthetic samples help to improve the classifier and ultimately affect its ability to decrease Bitcoin entity anonymity. Since our approach is based on balancing the initial population by adding new elements (resampling), we compare our GAN-based methodology with commonly used data-level techniques. GANs are typically implemented using two neural networks competing with each other in order to improve the ability of the whole system to learn and reproduce the input distribution. These "adversarial" models are mainly used in the domain of image processing [20] - currently being one of the most promising machine learning models in tasks related to image generation [21, 22].

Despite their potential, it is impossible to find a unique GAN solution that works for every scenario, which has led to the creation of different GAN architectures, each one with different goals and different application domains [23]. Therefore, one aim of this study is to investigate which type of GAN architecture can be used to generate synthetic Bitcoin address data behaviour and which architecture generates the most "valuable" information, i.e. valid synthetic samples that actually improve the Bitcoin entity classification. We also investigate how GAN training time affects classification results and whether results achieved with the best performing GAN setup are consistent when repeating the experiment. Finally, 5 state-of-the-art resampling techniques are used here and compared to our GAN approach in terms of classification performance and computational efforts.

To the best of our knowledge, this is the first work exploring the benefits and limitations of GANs in generating specific Bitcoin data and investigating in detail how the generated synthetic information affects Bitcoin entity classification.

The rest of the paper is organized as follows. In Section 2, concepts regarding the class imbalance problem and GANs as well as related work are introduced. In Section 3, the proposed solutions are presented, while Section 4 details the used data, the metrics and the experiments carried out in this study. In Section 5, results are reported and discussed and finally, Section 6, provides conclusions and guidelines for future work.

2 Preliminaries

In this Section, we recall the concepts behind Bitcoin classification and attacking its anonymity, resampling techniques to tackle class imbalance, and GANs. In particular, in Section 2.1, the Bitcoin entity classification for attacking Bitcoin anonymity is introduced. In Section 2.2, GAN architectures and their applications and limitations are described, finally, in Section 2.3, the most frequently used techniques to address class imbalance problems and how they are applied to Bitcoin data are reported.

2.1 Attack on Bitcoin anonymity

Bitcoin is the dominant cryptocurrency used in criminal activities as it allows non-transparent transactions and lacks effective regulatory mechanisms [24]. Over the years, the volume of transactions, cyber-attacks on specific Bitcoin entities [25] and illicit activities related to money laundering (ransomware, Ponzi schemes etc.) have increased [26], thereby promoting an increase in the usage of services that offer to protect user anonymity (like mixers [27]). Therefore, reducing anonymity within the network and classifying Bitcoin entities have become challenging and crucial tasks for Law Enforcement Agencies (LEAs) [5].

An entity is an actor in the Bitcoin network that controls or can control multiple public keys (i.e. Bitcoin addresses) and that does not always correspond to a single physical user (organization, corporation, small group of people). Entity classification represents an attack on Bitcoin's anonymity [28] as it allows the detection of entities that have a high risk of being involved in illicit transactions [29], as well as entities that are potentially more vulnerable to cyber-attacks. In fact, as described in [30], there are classes (like markets, ransomware, mixing, etc.) composed of entities more prone to making illicit transactions, while others are composed of entities that are more prone to be attacked (like exchanges, pool, etc.) because they manage a large amount of money [31, 32].

In Bitcoin entity detection, the idea is to exploit the information available in the blockchain, i.e. blocks and transactions, in order to define entity behaviours (or classes) and thereby classify entities [6].

These transactions can be used to extract valuable information linking input and output Bitcoin addresses, as well as other characteristics such as amount, fees, times etc. (Fig. 1). The available information provides a starting point for analysing the money flow but could be insufficient for defining the different entities. In this case, heuristics or external datasets are used. In the first case, addresses are clustered into entities following assumptions that represent common behaviours in the network [33], whereas in the second case, entity characterization is based on using external private or public Bitcoin "ground-truth" datasets [30, 34], which contain clusters of addresses belonging to known entities, the name of these entities and their related classes (Exchange, Gambling, Market, etc.).

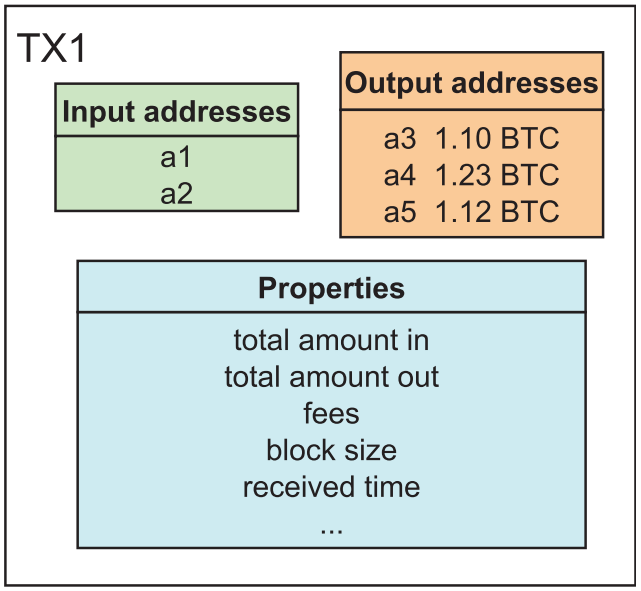


Fig. 1 Example of information that can be extracted from a Bitcoin transaction (TX1)

Combining Bitcoin transaction data with heuristics or external datasets allows one to obtain an *address-graph* in which each address is connected to a transaction and to other addresses that belong to the same entity, as shown in Fig. 2.

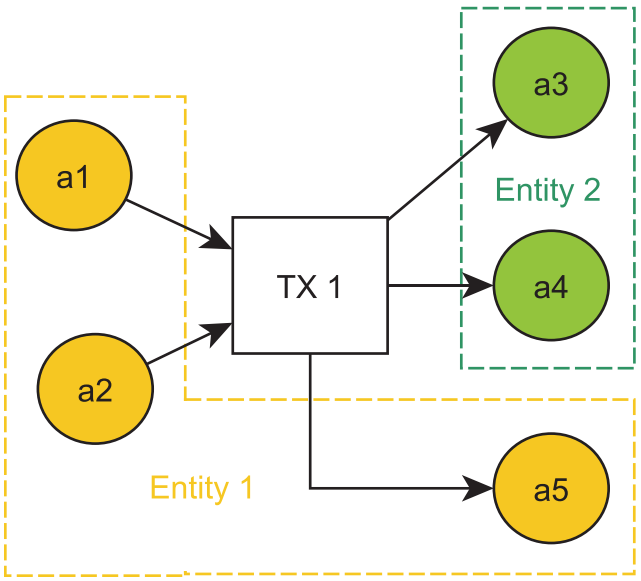


Fig. 2 Example of address-graph extracted using the Bitcoin transaction (TX1) data

The address-graph is the starting point for many Bitcoin entity classification studies [6, 29, 34, 35].

Entity behaviours are strictly related to the addresses used to define them, so when the classification is performed at the level of address information only (using input address features only), the operation is also called Bitcoin address classification [36]. Its aim is to define and predict entity behaviour (or class) associated with one or more addresses.

First classification approaches used statistical analyses and graph information as, for example, in [37], while recent works tend to exploit the power of machine learning techniques [29, 30]. This new trend makes it appealing to transfer technologies that have already shown good results in other domains to the Bitcoin blockchain domain. However, Bitcoin datasets typically do not contain homogeneous information regarding all Bitcoin entity classes (such as Exchanges, Mixers, Mining Pools, etc.), which means that there are more data pertaining to certain classes whereas others are underrepresented. This class imbalance problem represents a major obstacle during the training phase of machine learning models [8], since it dramatically affects the quality of classification results.

2.2 Generative adversarial networks (GANs)

A GAN is a generative model based on the joint optimization of two neural networks. Both networks represent players in a theoretical game designed to discover, learn, and replicate input distributions. The two neural networks are called Generator (G) and Discriminator (D), according to their tasks during the training phase. The objective of G is to learn the input distribution and generate synthetic samples similar to the real ones. The objective of D is to learn the difference between the synthetic and the real data evaluating the quality of G's samples. This competition drives both networks to improve their ability to learn from each other - creating a dynamic evolution of their neural parameters. The adversarial training ends when the optimization process stops, i.e. when the synthetically generated samples are indistinguishable from the real ones [19]. The first GAN, introduced in [19], follows the architecture presented in Fig. 3.

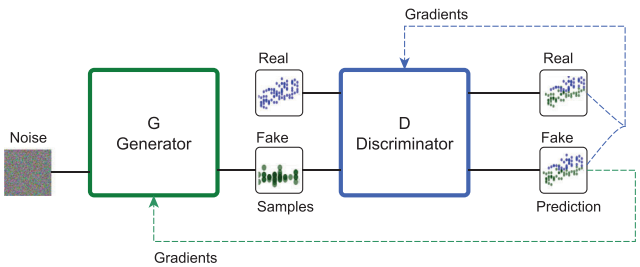


Fig. 3 General Generative Adversarial Network (GAN) architecture

Adversarial learning is characterized by a zero-sum non-cooperative game, also called *minimax* problem. The minimax function used in GAN implementations can be formulated with the parameterized networks G and D as introduced in [19] and reported in (1). $V(D, G)$ represents the value function in the two-player game, $D(x)$ is the discriminator's estimation of the probability that x (real data) is real, $E_{x\sim p_{data}(x)}$ is the expected value over all real data associated to the probability distribution of the real data $p_{data}(x)$, $G(z)$ is the generator's output (fake data) with noise input z, $E_{z\sim p_z(z)}$ is the expected value over all random inputs to the generator associated to a predefined prior noise distribution $p_z(z)$, and $D(G(z))$ is the discriminator's estimation of the probability that a fake sample is real.

$$\min_G \max_D V(D, G) = E_{x\sim p_{data}(x)}[\log(D(x))] + E_{z\sim p_z(z)}[\log(1 - D(G(z))] \quad (1)$$

Following (1), one network tries to maximize the effects of its actions during the training phase, while the second network tries to minimize its effects. Since the two networks are related via a common equation, improvements of one model worsen the other one, thus creating a dynamic learning system. As introduced by Goodfellow et al. [19], the goal of the training is to find a point of equilibrium between the two competing concerns, i.e. training converges when G and D reach the well-known Nash equilibrium [38]. A Nash equilibrium occurs when one player will not change its action regardless of what the opponent may do.

In order to find the equilibrium in the cost function, gradient descent optimization is used. This optimization updates both G and D simultaneously through stochastic gradient updates [39]. However, each scenario has its own suitable optimization function and there is no unique GAN architecture that works in every situation

GAN Limitations

Despite its learning abilities, the GAN architecture introduced in [19] comes with some problems that limit its usage compared to other architectures. Common collateral-effects [40], known as *non-convergence, vanishing gradients* and *mode collapse*, need to be taken into account at the time of implementation.

- *Non-convergence*: commonly occurs during the gradient descent optimization, where local minima and saddle points can stall the training, for example. As mentioned before, the goal of the training is to find an equilibrium in a game between two players, the generator G and the discriminator D. In a scenario where users "undo" each other's progress, no convergent solution may be reached, in which case oscillation in the models' parameters generate instability [41].

- *Vanishing gradient*: is a known problem affecting deep learning models that use gradient-based optimization. This phenomenon can produce slow training - for example when the solution follows a "pathological" curvature, which may even lead to non-convergence [42]. As presented in [43], the vanishing gradient can be produced when D is too good so that it does not provide enough information for G to learn the input distribution.
- *Mode Collapse*: also known as the Helvetica scenario, is produced when G learns to cheat D by generating only a limited variety of data regardless of the input. In this case, G learns just a small part of the input distribution that does not represent the entire population. In the worst case, the model "collapses", always generating the same sample. As presented in [41], mode collapse does not seem to be associated with any particular cost function.

Generally, there is no perfect solution to address all presented problems at once. However, previous studies tried to evaluate and mitigate some of these issues by using different GAN architectures, as proposed in Goodfellow et al. [41], for example, in which the authors showed that a modified minimax loss can be used to deal with vanishing gradients. Arjovsky et al. [44] preferred to use a Wasserstein distance as a loss function, creating the so-called Wasserstein GAN (or WGAN). The Wasserstein distance measures the distance between two probability distributions, in this case, the distribution of the data observed in the training dataset and the distribution belonging to the synthetic dataset. In this implementation, even when the supports of two distributions are located in two disjoint lower-dimensional manifolds, a smooth representation of the distance in-between is provided, which results in a better stabilization of the learning process using gradient descents. In fact, the WGAN presented in [44], showed vanishing gradient and mode collapse effects being drastically reduced. Yet, depending on the domain, WGANs can suffer from unstable training, slow convergence after weight clipping, and vanishing gradients. Local stability for both the WGAN and the traditional GAN can be guaranteed using an additional term during the gradient descent updates [45].

Another interesting architecture is based on unrolled GANs. In [46], it was shown how this technique solved the problem related to mode collapse and how it stabilized the training of GANs. In *unrolled GANs*, G not only considers the current discriminator information but also k future outputs of the discriminator versions in order to discourage G to exploit local optima. In particular, G will try to take steps that D will find difficult to respond to. For k steps, back-propagation occurs only to update a version of D's parameters (G's parameters being fixed) in order to allow D to optimize its performance, playing always against a specific G. The optimization is always performed through a gradient descent operation. Once the k steps are done, G's parameters are updated by back-propagating through all k steps ("unrolled" learning process).

In this paper, as in many other publications [47, 48], the GAN architecture introduced in [19] is called *Vanilla* GAN. Two more GAN architectures (Wasserstein [44] and unrolled GAN [46]) are presented and implemented in the following Sections.

GAN for Cybersecurity

The main goal of these adversarial networks is to approximate the real data distribution and generate synthetic samples for enhancing/enriching the original dataset [49] - for example for creating infrared high-resolution images [50, 51], realistic vehicle images [52], for enhancing underwater images [53], for image inpainting [54] and for palmprint recognition [55]. In the speech domain, GANs are used to compute an enhancement operation [56], to improve Neural Machine Translation (NMT) results generating human-like translations [57] and for emotion recognition by creating synthetic audios from audio-visual datasets [58].

Only a few recent studies apply GANs in the context of cybersecurity. In [59] and [60], GANs are used to generate new cyber-attack samples from existing data. In the former, the goal is to balance the initial dataset and improve intrusion detection systems; in [60] the objective is to train a binary classifier (attack, no-attack) and show the benefits in terms of accuracy and f1-score generated by the balanced dataset. Mukhtar et al. [61] propose an approach based on GANs and Siamese networks for generating synthetic data of side-channel attacks. In [62], a new GAN-based framework is introduced to address the class imbalance in an encrypted traffic dataset and is compared to models trained with balanced datasets using SMOTE, ROS, and Vanilla GAN techniques.

GAN architecture and its parameters always need to be chosen carefully, depending on the given scenario. For this reason, in this work, a wide variety of optimization heuristics and GAN architectures have been compared with the aim to detect the architecture that generates "highly-valuable" synthetic samples. The "value" of synthetic samples is evaluated by analysing the improvements they generate in the classification results.

2.3 The class imbalance problem

In machine learning, the information available for describing a problem at hand is a key factor and the generation of a machine learning model is strongly related to the

number of observations. One or more categories (or classes) in the initial dataset having more samples than others may significantly affect the training phase, generating phenomena called the class imbalance problem [63]. In this situation, supervised machine learning systems tend to be "overwhelmed" by the majority class, making it hard to discover robust patterns for under-represented classes.

As we have already mentioned, the common algorithms used for addressing class imbalance can be grouped into 4 categories. Among them, algorithm-level and cost-sensitive methods are usually more dependent on the problem and ad-hoc solutions could be required. In neural networks, considering costs may be straightforward [12] and may yield similar results as data-level techniques (e.g. random oversampling) if weights are assigned for balancing the importance of the classes. Otherwise, data-level techniques tend to be more versatile, since they are independent of the classifier used, acting directly on data. In this sense, they provide more diversity in the samples, since new synthetic data may be generated (e.g. SMOTE or ADASYN) favoring the learning in neural networks.

In this work, we are interested in analysing *data-level techniques*, which are among the most used strategies [64]. These approaches are based on resampling and thus allow us to compare their results with our GAN-based approach. In particular, data-level techniques are categorized into *over-sampling* or *under-sampling*. In the first case, data are added to the less populated classes in order to reach the same (or similar) number of elements as the majority class. In the second one, data are removed from the majority classes in order to reduce the number of samples down to the same (or similar) amount of elements that describe the least represented class. Both sampling strategies can be combined creating hybrid methods.

The usage of these strategies mitigates the problem related to unbalanced data but can produce downside effects in the supervised machine learning model [8]. For example, in under-sampling, the simplest technique (Random Under Sampling or RUS) involves removing random records from the majority class, which can cause a loss of information. The simplest implementation of over-sampling (Random Over Sampling or ROS) duplicates random observations from the minority class, which, in turn, can cause overfitting. These problems are addressed by implementations of other - more complex - resampling techniques such as Tomek Links (TL) [65], Synthetic Minority Over-sampling Techniques (SMOTE) [66] and Adaptive Synthetic (ADASYN) [67] approaches.

TL is based on a heuristic approach that uses an enhancement of the Nearest-Neighbor rule. Even though it is considered to be an under-sampling strategy (as it removes Tomek links), its result is not a uniform dataset with the same number of elements for each class.

SMOTE is an over-sampling technique that creates new instances between minority class samples that are close to each other in the feature space [66]. Typically, the algorithm first computes the k-nearest neighbors for a sample of the minority class and then creates the synthetic instance by randomly choosing one point in the line segment that connects one of the k neighbors with the considered sample. The process is repeated until a degree of balance is reached. The number of points generated for each data sample is uniform, uniquely based on the chosen k.

ADASYN was implemented with the aim of improving the results obtained via the SMOTE strategy [67]. The main difference between them is how to decide the number of synthetic samples generated for a particular point: in ADASYN distribution is computed for each point, whereas in SMOTE a uniform distribution is used.

More complex approaches for addressing the imbalance problem are based on generative learning, which can be categorized into two groups [14]: traditional generative models and deep generative models. The first ones are based on traditional machine learning algorithms that usually use probability/density functions for approximating the input distribution, while the second models are based on deep learning algorithms, which allow them to learn and generate more complex distributions. For this reason, deep learning models represent more challenging and interesting structures. Well-known examples are the deep Boltzmann machine (DBM), deep belief networks (DBN), variational autoencoder (VAE) among others. More specifically, DBM and DBN are energy-based models, which means that the joint probability is defined using an energy function. In particular, for both approaches, their components are trained separately and then joined in a separate phase [68]. In this sense, the resulting generative model is more difficult to stack without causing a quality loss. On the other hand, VAE and GAN models are easier to be trained. However, although VAEs facilitate the comparison among different implementations, their performances are strongly conditioned by the reconstruction error, which can generate "blurred" samples. On the other hand, GANs allow one to learn more complex distributions and efficiently generate more realistic samples, even though they have some limitations as described in Section 2.2. It is to be noted that generative models not only increase the representation of minority classes, but by learning the entire data distribution they are able to generate more variable data. This broad generation mitigates effects like underfitting and overfitting, which are challenging problems for traditional resampling techniques like RUS and ROS. Furthermore, for other resampling techniques like SMOTE or ADASYN, synthetic samples are based on local information (neighbors) only, without taking into account the overall distribution [69]. For this reason, GAN sample generation can be seen

as an "intelligent over-sampling" in which complex and high dimensional behaviours are created by analyzing all available and not just partial information.

Inspired by the promising results that GANs have demonstrated in computer vision and natural language processing tasks [50, 58], in this work, we present an approach that uses such deep generative models to address the class imbalance problem related to Bitcoin entity behaviour classification. More specifically, the ability of GANs to learn and generate data by analysing the overall data distribution and not only local information, could have a strong impact on scenarios in which entities evolve over time as it is the case in the Bitcoin domain. In these scenarios, reduced local information may be related to old and no longer relevant entity behaviour. Furthermore, learning from the overall distribution allows GANs to generalize Bitcoin entity behaviours. This generalization is useful not only for exploring more deeply the feature space and for increasing classifiers' abilities, but could be used by the community and LEAs for investigating novel entity patterns that have not yet been detected in the Bitcoin mainnet.

Class imbalance strongly affects the quality of classification, and it represents a relevant problem associated with datasets such as the Bitcoin blockchain. In particular, in [70], the class imbalance problem is not considered, and several supervised machine learning models are implemented with the aim to identify 16 licit-illicit categories of users. Authors conclude their work highlighting that due to limited instances of classes in the training data several legal and illegal classes were misclassified. Two methods frequently used in fraud detection to tackle dataset imbalance, cost-sensitive [10] and sampling-based [11] approaches, are leveraged in [71] to address the Bitcoin class imbalance problem while implementing a machine learning model for detecting Bitcoin Ponzi schemes. In [72] and in [73], authors implement models able to detect if an address belongs to an Exchange (binary classification), resolving the class imbalance problem using RUS. In [72], a sampling technique is used over the transaction-directed hypergraph, while in [73] a sampling technique is used for removing some nodes and for guaranteeing the balance between elements of two considered classes (Exchange and not-Exchange). In [36], a model for analysing Bitcoin addresses aiming to detect abnormal activities is built, and the class imbalance problem is managed by using stratified random sampling. Harlev et al. [7] applied SMOTE in order to enrich the original dataset and train a supervised machine learning model for predicting entity classes. The model trained with the SMOTE-resampled dataset showed slight improvements in terms of overall accuracy and f1-score. These improvements were limited as although SMOTE generated improvements in f1-score values for the initial minority classes, at the same time, it decreased the values of other considered classes. Monamo et al. [74] use unsupervised machine learning (k-means) for detecting Bitcoin frauds. They implement z-scores, chord distance, and Hellinger distance with bagging and boosted classifiers in order to describe the input dataset and compare the results with a SMOTE-resampled dataset. The latter implementation outperformed all the considered methods generating improvements for minority classes in terms of sensitivity. Nevertheless, according to the authors, this oversampling technique decreased the reliability and increased the number of false positives.

Recently, GANs have been used for forecasting Bitcoin market prices [75] or for analysing Bitcoin price trends and the stock market [76]. In [77], we study the best parameters to be used for training adversarial algorithms for learning Bitcoin data. However, generation was limited to one specific class only using a Vanilla-GAN implementation, and without evaluating them for improving the classification task. Han et al. [78] used conditional Wasserstein GAN (WCGAN), Deep Regret Analytic (DRAGAN), and SMOTE technologies for addressing Bitcoin imbalance to perform a binary classification, i.e. distinguish legitimate and illegitimate transactions. However, they used a limited dataset gathered from a single marketplace (named Silkroad) closed in November 2014, composed of 16 distinct features. The authors acknowledged that the limited dataset and the fact that its distribution is clearly concentrated on certain values could promote the Mode Collapse effect. Furthermore, despite testing different GAN architectures, they did not achieve clear improvements in the (binary) classification task.

Inspired by the results presented in [77, 78], in this work, we present a detailed analysis of three different GAN architectures and their adaptation for addressing the Bitcoin class imbalance problem. Then, we evaluate how the generated synthetic samples improve the Bitcoin entity classification (multi-class problem). Finally, the best results obtained with GAN-resampled datasets, are compared with other state-of-the-art resampling strategies, in order to evaluate the benefits and limitations of the proposed methods.

3 Proposal and contributions

In this work, we present a solution based on GAN architectures for addressing the Bitcoin class imbalance problem to improve entity classification and consequently decrease Bitcoin entity anonymity. The idea is to use the adversarial technique to model the real data distribution and then perform data augmentation through synthetic data generation (over-sampling strategy). As introduced in

Section 2.3, these generative models allow to obtain a more variable dataset, which not only avoids traditional downside effects (under-sampling and over-sampling), but also helps in the generalization of the entity behaviour.

In particular, our idea is that for each under-represented (minority) class, i.e. for all the Bitcoin classes excluding the most represented class, an adversarial model is trained. This training is performed by considering each class separately. Then, each GAN is used to generate sufficient synthetic data so that each class reaches the population size of the majority class. This operation creates an enriched hybrid dataset (formed of synthetic and real data) used to train the Bitcoin classifier, which is finally tested in order to evaluate how the synthetic samples have affected entity anonymity.

In this study, we implement three different GAN architectures: the Vanilla GAN, the Wasserstein GAN (WGAN) and the unrolled GAN. In particular, the first architecture helps us validate how possible GAN downside effects affect the training of a simple GAN when Bitcoin behavioural data are used. The second architecture (WGAN) is chosen for its ability to improve the model stability and make the training process easier [44]. Finally, the third architecture (unrolled) is chosen as it allows one to improve the GAN dynamics by bringing the discriminator closer to an optimal response [46]. We think that these three chosen GAN architectures represent a good benchmark set for evaluating the benefits and limitations of such technology when applied to the Bitcoin domain.

These GAN architectures are studied and implemented in order to determine the most suitable for the Bitcoin domain, which can be used for creating an enriched version of the initial dataset.

Each architecture is evaluated in several checkpoints, i.e. using different GAN training times in terms of epochs. This operation allows us to evaluate how the generation of synthetic samples is related to GAN training epochs, and when they are affected by GAN downside effects mentioned above (overfitting). In each checkpoint, the respective data generated by GAN Generators are used to train a distinct version of the address classifier.

From this point onward, we indicate an address classifier that has been generated based on data enriched with synthetic samples produced by a GAN trained up to the $i-th$ checkpoint, as $GAN\text{-}classifier_i$.

We repeat the implementation of the best performing GAN architecture 5 times. This approach helps us to validate the mean and standard deviation of the classifier's metrics in order to check its repeatability and robustness when a GAN-resampled dataset is used. Finally, the best performing GAN-based classifier is compared with classifiers using data generated with other state-of-the-art resampling strategies.

Our main contributions can be summarized as follow:

1. Adapting state-of-the-art GAN technologies in order to learn Bitcoin behaviours;
2. Evaluating how synthetic Bitcoin samples generated by GANs improve Bitcoin entity classification;
3. Comparing three different GAN architectures in terms of generated classification improvements;
4. Studying how training time affects the GAN learning process, the synthetic data generation and the final Bitcoin entity classification;
5. Analysing the repeatability of the best GAN approach, repeating the execution 5 times to evaluate metrics' means and standard deviations;
6. Comparing the GAN-based approach with 5 traditional data-level techniques in terms of obtained Bitcoin classification performance and computational costs.

4 Experimental framework

In this Section, we will first provide an overview of the data used in this study (Section 4.1), then, in Section 4.2, preprocessing steps are presented and in Section 4.3 GAN configurations are explained. In Section 4.4, the metrics used for evaluating the models are listed and in Section 4.5, the idea behind each experiment is described.

4.1 Dataset

For this study, we used information obtained from the Bitcoin mainnet and the WalletExplorer[1]. The whole Bitcoin blockchain was downloaded by using the Bitcoin Core[2], and in particular all blocks and transactions from the beginning until block number 570,000 were downloaded, corresponding to blocks mined until April 3rd 2019, 09:20:08 AM.

At the same time, we downloaded a labelled address-entity dataset available on the website WalletExplorer. This platform contains information about transactions, addresses and real-world entity names detected over the years. Their databases are continuously updated and thus have been used as "ground truth" for many Bitcoin-related studies, as in [29, 79]. In this study, we considered six entity classes:

- *Exchange*: entities that allow their customers to trade among cryptocurrencies or to change cryptos for fiat currencies (or vice-versa);
- *Gambling*: entities that offer gambling services based on Bitcoin currency (casino, betting, roulette, etc.);

[1] https://www.walletexplorer.com/

[2] https://bitcoin.org/en/download

- *Mining Pool*: entities composed of a group of miners that work together sharing their resources in order to reduce the volatility of their returns;
- *Mixer*: entities that offer a service to obscure the traceability of their clients' transactions;
- *Marketplace*: entities allowing to buy any kind of goods or services using cryptocurrencies. Some of them potentially related to illicit activities;
- *Service*: entities that allow users to lend Bitcoins and passively earn interests, or allow them to request a loan.

As shown in Table 1, 327 different entities and more than 16,000,000 addresses were downloaded from WalletExplorer. Exchanges represent the most populated class - its samples represent more than 44% of the entire entity dataset and about 62% of the address dataset. On the other hand, Mining Pool is the least populated class in terms of addresses with a ratio of 0.53% and the Market class is the minority in terms of entities (about 6%). This overview highlights the class imbalance problem associated with Bitcoin datasets.

Bitcoin blockchain and WalletExplorer data were combined in order to obtain a labelled address dataset, fundamental for a supervised machine learning task. This new dataset was used as a starting point for extracting the features used to define address behaviours and subsequently to create the address dataframe, as shown in Fig. 4. The address dataframe was created following concepts presented in [29], and extracting 7 features related to known Bitcoin addresses: the number of transactions in which a certain address is detected as receiver/sender, the amount of Bitcoin (BTC) received/sent from/to this address, the balance, uniqueness (if this address is used in one transaction only) and siblings.

As shown in Fig. 4, the address dataframe was spilt into training and testing datasets with a proportion of 50/50 keeping class distributions unchanged (stratified).

4.2 Data preprocessing

Analysing the training dataset and the dependence among its available features, it was possible to perform a reduction

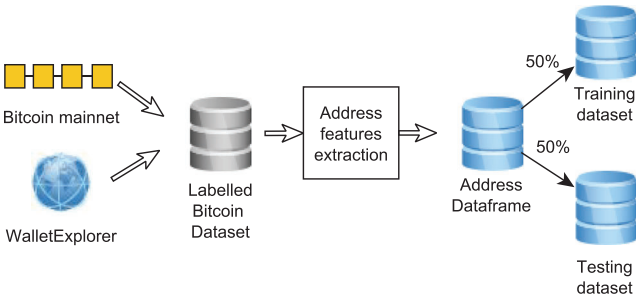


Fig. 4 Address dataset creation

of the dataset dimensionality. The total amount of BTC received, the total amount of BTC sent and the balance are non-independent variables, so we decided to train the GAN such that it only learns two of them, whereas the third one was calculated. Thereby, the GAN learned the distribution of the amount of BTC received and the balance. The amount of BTC sent was computed a posteriori. In the same way, the uniqueness and the total received transactions are related, since one address is unique (1) when it is used exactly once for receiving money, otherwise, it is not unique (0). Following this rule, the total of received transactions was used to train the GAN, and the uniqueness values were computed a posteriori. In this manner, the 7 initial features were reduced to 5.

The training dataset was split separating the samples of each class, obtaining 6 different datasets. Each class dataset was then used for training a distinct GAN, except for the (highly populated) Exchange dataset (Fig. 5). Before training, each class dataset was normalized to reduce GAN learning complexity by limiting the feature distributions in a fixed range between 0 and 1. Let F^m be the set of features that characterize each class dataset, $\forall f \in F^m$, i.e. for each specific feature, the normalization was performed following (2), where x are elements in f, $X_{max} = \max(f)$, $X_{min} = \min(f)$, and $\widetilde{x}$ represents the normalized value. Once the GANs were trained and synthetic samples could be generated, they were de-normalized using the (3) and, after computing the non-independent features, they were added to the initial (real) data, creating an Hybrid dataset, as shown in Fig. 6. The de-normalization was performed

Table 1 Overview of the used WalletExplorer (WE) data (the overall values are in bold)

Class	# Entity WE	% Entity WE	# Address WE	% Address WE
Exchange	144	44.04	9,947,450	61.56
Gambling	76	23.24	3,050,899	18.88
Marketplace	20	6.12	2,349,111	14.54
Mining Pool	27	8.26	85,887	0.53
Mixer	37	11.31	475,781	2.94
Service	23	7.03	250,788	1.55
Total	**327**	**100**	**16,159,916**	**100**

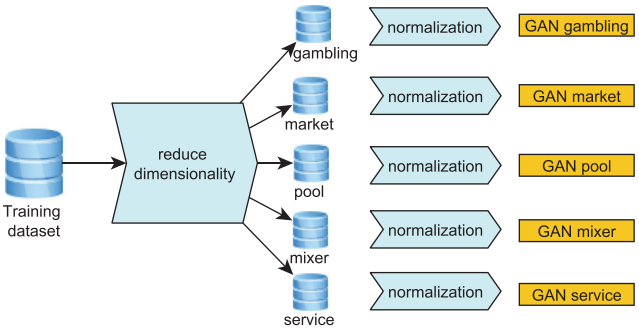


Fig. 5 Dimensional reduction, class separation and normalization for GAN training

in order to work directly with Bitcoin data in their real space.

$$\widetilde{x} = \frac{x - X_{min}}{X_{max} - X_{min}} \tag{2}$$

$$x = [(X_{max} - X_{min}) \times \widetilde{x}] + X_{min} \tag{3}$$

4.3 GAN implementation

Despite using three different GAN architectures in this paper, the Generator (G) and the Discriminator (D) networks were the same. In [44], when working with images, the authors suggest using a 4-layer Neural Network for both G and D, where each layer is defined by 512 neurons. In our case, since the size of the Bitcoin feature space is lower than the ones used for the images, three hidden layers for both G and D were implemented, following the specifications described in [77]. In particular, G's neural network was composed of three hidden layers with 512, 256 and 128 neurons (Fig. 7), all using the Rectified Linear Unit (ReLu) as activation function [77]. The output was fixed to 5, i.e. the same number of non-independent features in the real samples. A technique called Root Mean Square Propagation (RMSProp [80]) was used in all implementations to optimize the relative cost function with a learning rate set to $5e-5$, as indicated in [44].

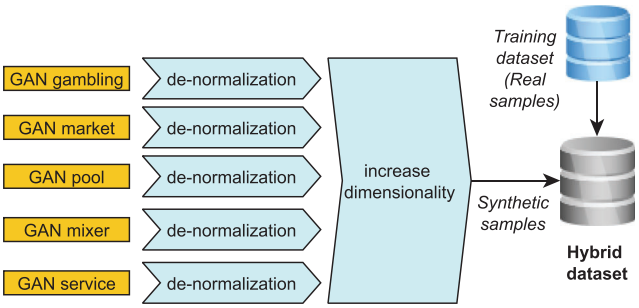


Fig. 6 Creation of the hybrid (augmented) dataset

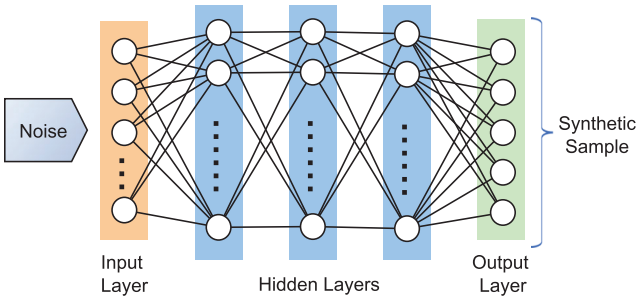


Fig. 7 Generator architecture

The value of the batch size, representing the number of elements used at once for updating the weights of the neural networks, was kept fixed to 1,000 samples. Furthermore, for the unrolled GAN, the k value of the steps forward was chosen to be equal to 5, as used in several tests in the introduced paper [46].

D's neural network was also composed of three hidden layers with 256, 512 and 256 neurons (Fig. 8), all again using the Rectified Linear Unit (ReLu) as activation function [77]. As shown in Fig. 8, the input size of the Discriminator was 5, i.e. equal to the number of non-independent real features as well as to the number of synthetically generated features.

For each architecture, 6 GAN training time checkpoints were fixed a priori. These values represent the training length of the adversarial networks, indicated in epochs. In particular, checkpoints were fixed in: 1,000, 10,000, 25,000, 50,000, 75,000 and 100,000 epochs.

4.4 Evaluation metrics

This Section describes the classification metrics used to evaluate and compare the different machine learning models in our experiments. Assuming N to be the total number of classes and for each class i assuming tp_i as true positive value, fp_i as false positive value, tn_i as true negative value, fn_i as false negative value, the following metrics were defined.

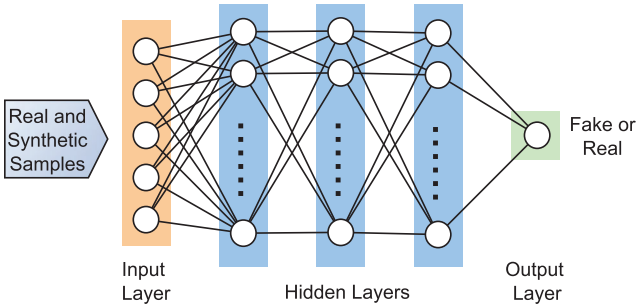


Fig. 8 Discriminator architecture

- *Accuracy* or *Score* is defined as the number of correct predictions divided by the total number of predictions and is given as percentage (4).

$$\frac{\sum_{i=1}^{N} \frac{tp_i + tn_i}{tp_i + fn_i + fp_i + tn_i}}{N} \tag{4}$$

- *Precision (prec.)* is the number of positive predictions divided by the total number of the positive class values predicted. It represents a measure of a classifier's exactness given as a value between 0 and 1, with 1 relating to high precision (5).

$$\frac{\sum_{i=1}^{N} tp_i}{\sum_{i=1}^{N} (tp_i + fp_i)} \tag{5}$$

- *Recall or Sensitivity* represents a measure of a classifier's completeness quantifying the number of positive class predictions made over all positive examples in the dataset ((6)). It is given as a value between 0 and 1

$$\frac{\sum_{i=1}^{N} tp_i}{\sum_{i=1}^{N} (tp_i + fn_i)} \tag{6}$$

- *f1-score* is the harmonic mean of Precision and Recall. It takes values between 0 and 1, with 1 relating to perfect Precision and Recall and can be calculated using (7)

$$F_1 score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{7}$$

- *Matthews Correlation Coefficient (MCC)* is a metric yielding easy comparison with respect to a random baseline, particularly appropriate for unbalanced classes. It takes values between -1 and $+1$. A coefficient of $+1$ represents a perfect prediction, 0 an average random prediction and -1 an inverse prediction. As shown in [81], let K be the number of classes and C be a confusion matrix with dimensions $K \times K$, the MCC can be calculated as shown in (10)

$$MCC_{p1} = \sqrt{\sum_{k} \left(\sum_{l} C_{kl} \right) \left(\sum_{f,g|f \neq g} C_{gf} \right)} \tag{8}$$

$$MCC_{p2} = \sqrt{\sum_{k} \left(\sum_{l} C_{lk} \right) \left(\sum_{f,g|f \neq g} C_{fg} \right)} \tag{9}$$

$$MCC = \frac{\sum_k \sum_l \sum_m C_{kk} C_{lm} - C_{kl} C_{mk}}{MCC_{p1} \times MCC_{p2}} \tag{10}$$

- *Area Under the Receiver Operating Characteristic Curve (AUC)* measures the two-dimensional area underneath the ROC curve, which is a plot of the true positive rate against the false positive rate. It indicates the classifier's ability to avoid false classification and it takes values between 0 and 1, with 1 relating to perfect predictions. AUC can be calculated for a binary classification as described in (11). This equation can be extended for multi-class problems with some adjustments.

$$AUC = \frac{1}{2} \left(\frac{tp}{tp + fn} + \frac{tn}{tn + fp} \right) \tag{11}$$

Furthermore, the term *avg.* is used in the following to indicate average results of an indicated metric and the term *std.* is used to indicate the computed standard deviation.

4.5 Overview of the planned experiments

In this study, four experiments were carried out in order to explore the GAN approach for addressing the Bitcoin class imbalance problem. The obtained results were then compared with current state-of-the-art techniques in order to validate and check the limitations of the introduced methods.

The first experiment introduces the baseline classifier by directly using the training dataset obtained from the address dataframe (Fig. 9). The creation of this baseline model not only was used for detecting limitations in the Bitcoin entity classification but was also used later for evaluating improvements achieved by other presented experiments involving resampled data. The baseline model was implemented using a Random Forest model. Following our previous study on Bitcoin classification [29], the number of estimators was set to 10, and the Gini function was used to measure the quality of the split without fixing a maximum depth of the tree. The testing dataset was used to compute a first evaluation of how the baseline classifier (trained with real data) predicts entity classes related to a certain address.

In the second experiment, our solution based on adversarial learning through GANs was implemented in order to generate synthetic Bitcoin address behaviours. The experiment started by training GANs based on the training dataset. In particular, as described in Section 4.2, a single GAN was implemented for each class that was underrepresented in the training dataset (Gambling, Market, Mining Pool, Mixer, and Service). Following this specification, the GAN trained using on the largest dataset was the GAN for generating synthetic Gambling data with 1,527,247 real samples, while the GAN using the smallest dataset was the Mining Pool GAN based on 43,265 real samples only (Table 2).

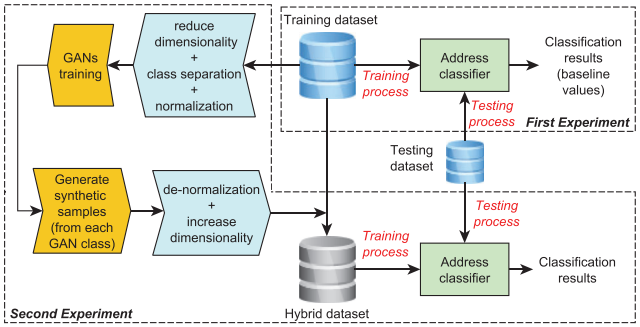


Fig. 9 Schema of the first and second experiment

At the end of the training process, the Generators G of each class GAN were used to create new synthetic samples that were then de-normalized and joined with the real training dataset (after computing the 2 dependent features), creating the hybrid dataset (Section 4.2). Finally, this hybrid information was used to train an address classifier (Fig. 9), which was based on Random Forest models with the same setup as in the first experiment. Once this classifier was ready, it was tested using the testing dataset. This experiment was carried out by stopping the GANs' training processes in 6 different checkpoints (epochs), as explained in Section 4.3. In each one of them, the Gs were used to enrich the real dataset creating different versions of the hybrid dataset, thus creating a specific version i of the GAN-classifiers (address classifiers).

Algorithm 1 Pseudo-algorithm followed to evaluate each GAN architecture.

```
Split the initial dataset per class;
Normalize each class dataset;
Reduce dimensionality of each class dataset;
for i in checkpoints do
    Training a GAN for each class dataset;
    Generate synthetic samples (until each minority class reaches a population comparable to the majority one);
    De-normalize synthetic data and increase the dimensionality;
    Enrich the training dataset with synthetic data
    Train GAN-classifier_i with the new enriched dataset;
    Test GAN-classifier_i with an unseen dataset (testing dataset);
end for
```

The steps presented in Algorithm 1 were repeated using the three types of GAN architectures, Vanilla GAN, Wasserstein GAN (WGAN), and unrolled GAN. The latter two techniques were used in order to evaluate and mitigate possible collateral effects that typically affect adversarial networks.

In the third experiment, the most promising solutions detected in the second experiment were analysed in detail. In particular, only architectures and checkpoints that had shown the highest classification values were used. For this configuration, the GAN training and the classifier training were repeated 5 more times, in order to check the repeatability of our results and avoid outlier solutions. In the fourth and last experiment, the classification results obtained in the third experiment were compared with classification results obtained via several known resampling techniques (Section 2.3). In this experiment, two under-sampling techniques, RUS and TL, and three over-sampling techniques, ROS, SMOTE, and ADASYN, were used. These resampling techniques were directly applied to the training dataset in order to use this enhanced dataset to train the Bitcoin entity classifier. The subsequent model was again a Random Forest classifier with the setup from the previous experiments and was tested using the same testing dataset.

The fourth experiment allowed us to determine the suitability of current techniques for addressing class imbalance problems in the Bitcoin domain and allowed us to find insights about how these techniques affect Bitcoin entity behaviour classification. Furthermore, comparing resampling techniques with our GAN approach highlighted the strengths and weaknesses of applying GANs to Bitcoin data.

5 Experimental study

In this Section, the four experiments with their obtained results are presented. In particular, in Section 5.1, the baseline model is built, in Section 5.2, the three GAN architectures are implemented and compared. Then, in Section 5.3, the best GAN solutions are used to check the repeatability of the results, whereas in Section 5.4, the GAN approach is compared with 5 resampling techniques. Finally, in Section 5.5, results are discussed.

5.1 Baseline model

Table 2 provides an overview of the results obtained by testing the baseline model. In particular, the number of real samples used for training the baseline model is reported as well as the values for precision, recall and f1-score per class, an overall average of these three metrics and the overall accuracy obtained using the entire testing dataset.

The baseline model showed an overall good accuracy value of 94.67%, however, it presented problems in detecting samples belonging to minority classes, which suggests

Table 2 Baseline classifier accuracy and f1-score obtained with the testing dataset (the average values are in bold)

Class	# train samples	prec.	recall	f1 score
Exchange	4,972,019	0.95	0.98	0.96
Gambling	1,527,247	0.93	0.89	0.91
Marketplace	1,174,468	0.99	0.97	0.98
Mining Pool	43,265	0.89	0.75	0.82
Mixer	237,725	0.86	0.80	0.83
Service	125,531	0.84	0.67	0.75
avg.		**0.91**	**0.84**	**0,87**
score %	**94.67**			

that the class imbalance problem is affecting the classification of minority classes. While majority classes were detected with high f1-score values over 0.90 (Exchange, Gambling and Marketplace), less represented classes (Mining Pool, Mixer, and Service) yielded respectively lower values of 0.82, 0.83 and 0.75.

The f1-score values were conditioned by lower recall values, indicating problems in detecting underrepresented classes. For example, among all the Service samples in the testing dataset, only 67% (i.e. recall 0.67) and only 75% and 80% of the Mining Pool and Mixer elements were correctly classified.

The obtained results underline again the importance of having a balanced dataset in multi-class classification problems using supervised machine learning. Results were affected by under-represented classes in the original Bitcoin dataset.

5.2 Comparison among GAN architectures

GANs were used for generating synthetic data to enrich the training dataset and create a hybrid dataset. The number of synthetic samples generated by each GANs is correlated with the class population of the initial training dataset (Table 2). The idea is to balance the initial class distributions by adding a sufficient amount of synthetic samples for each minority class. In particular, as shown in Fig. 10, the most populated class - Exchange - has 5,000,000 real samples, hence we chose to generate 3,000,000 synthetic samples for the Gambling class, 3,500,000 synthetic samples for the Market class, and 4,500,000 synthetic samples for Pool, Mixer, and Service class, respectively for each checkpoint. In this manner, the new hybrid dataset contained a new per-class distribution varied in a range between 4,527,247 (Gambling class) and 4,972,019 (Exchange class).

Once the hybrid dataset belonging to a specific checkpoint (epoch) was generated, it was denormalized and the dependent features were computed. This new dataset was used to train the GAN-classifier following the same process used for creating the baseline model. This GAN-classifier was then tested with the testing dataset, in order to evaluate its global accuracy and precision, recall, and f1-score for each class. This comparison over the same testing dataset allowed us to evaluate how much the new synthetic information generated by the GANs affected the entity classification and consequently how much the GANs have learned from the real distribution.

Results regarding the Vanilla GAN architecture are reported in Table 3; in particular the values for f1-score, precision and recall obtained from each GAN-classifier and for each class are indicated. These metrics improved with respect to the ones obtained with the baseline model, as well as the overall accuracy and the average f1-score. The two best solutions, GAN-classifier$_1$ and GAN-classifier$_3$, achieved by training the GANs with 1,000 and 25,000 epochs, in terms of overall recall reaching both values of 0.90 and in terms of f1-score as well, which was, respectively 0.05 and 0.06 points higher than the baseline values. Improvements were also visible analysing the metrics of each class individually. Even the majority class (Exchange), which was not affected by the resampling strategy, increased its f1-score by 0.02 for the two best solutions. From the results shown in Table 3, one can observe that classifiers implemented with the hybrid dataset

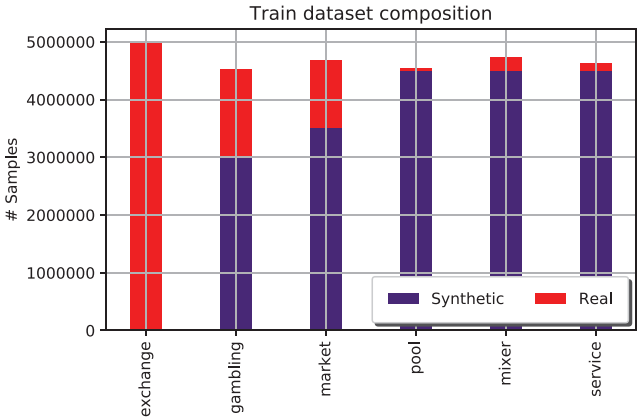


Fig. 10 Synthetic and real sample distribution in the newly created hybrid dataset

generated by the Vanilla GANs which were trained with a low number of epochs (1,000 and 25,000) performed much better than the other ones.

The same study as presented for the Vanilla GAN was carried out by using a WGAN architecture. The obtained results are reported in Table 4. They showed a different trend, as the classifiers with the best global f1-score were obtained by creating the hybrid dataset using GANs that were trained longer, i.e. the ones trained in 50,000, 75,000 and 100,000 epochs. These three best options achieved an average f1-score of 0.90. However, the WGAN approach presented problems in detecting Mining Pool samples. In fact, although improvements in terms of recall were generated, precision values dropped, causing a decrease of f1-scores related to that class (about 0.2 - 0.3 below the baseline result).

In Table 5, computed evaluation metrics for the unrolled GAN architecture are reported. In this case, results tended to follow the trend of the Vanilla GANs, where best solutions were obtained with GANs that were trained less (in terms of epochs), yet without reaching high scores. The best solutions were obtained by using GAN-classifier$_1$ and GAN-classifier$_3$ (unrolled GAN trained for 1,000 and 25,000 epochs), showing the highest values in terms of precision, recall and f1-score.

As can be seen from Table 6, all GAN solutions show higher values in terms of AUC (0.98-0.99) regardless of the chosen architecture and checkpoint. On the other hand, both accuracy and MCC follow the trend of the averaged f1-score shown in Tables 3, 4, and 5. The Vanilla GAN shows the best accuracy and MCC for models trained respectively with 1,000 and 25,000 epochs, similar to the unrolled GAN, whereas the WGAN shows the highest accuracy and MCC values in 50,000, 75,000 and 100,000 epochs.

Comparing the average recall and f1-scores achieved with all GAN-classifiers implemented with the three proposed GAN architectures, we found that the best solutions with the highest improvements in Bitcoin entity classification were obtained using the Vanilla GAN architecture. In fact, the Vanilla GAN yielded important improvements in terms of minority class detection, as shown by high values of recall and f1-scores.

In Fig. 11, the overall accuracy of the GAN-classifiers implemented using the three GAN architectures are compared. The accuracy of the GAN-classifiers trained with data generated from the Vanilla GAN model was usually higher than the accuracy values obtained with the classifiers trained based on data generated by the other two GAN architectures. However, moving to longer epochs, the downward trend in terms of accuracy that we observed may be a symptom of mode collapse: potentially, the GAN stopped learning the real distribution and started generating similar samples only, which did not contribute

Table 3 Precision, recall and f1-scores obtained with the GAN-classifiers at different epochs (training dataset enriched with synthetic samples generated by the Vanilla GAN). Average values are in bold

	1,000 epochs (GAN-classifier$_1$)			10,000 epochs (GAN-classifier$_2$)			25,000 epochs (GAN-classifier$_3$)			50,000 epochs (GAN-classifier$_4$)			75,000 epochs (GAN-classifier$_5$)			100,000 epochs (GAN-classifier$_6$)		
Class	prec.	recall	f1 score	prec.	recall	f1 score	prec.	recall	f1 score	prec.	recall	f1 score	prec.	recall	f1 score	prec.	recall	f1 score
Exchange	0.97	0.99	0.98	0.96	0.98	0.97	0.96	0.99	0.98	0.96	0.99	0.97	0.96	0.98	0.97	0.95	0.98	0.97
Gambling	0.97	0.93	0.95	0.95	0.91	0.93	0.96	0.92	0.94	0.96	0.92	0.94	0.95	0.91	0.93	0.94	0.90	0.92
Market	0.99	0.98	0.99	0.99	0.97	0.98	0.99	0.98	0.98	0.99	0.98	0.98	0.99	0.97	0.98	0.99	0.97	0.98
Pool	0.82	0.83	0.82	0.77	0.81	0.79	0.94	0.84	0.89	0.92	0.82	0.87	0.91	0.79	0.85	0.90	0.77	0.83
Mixer	0.91	0.86	0.88	0.92	0.85	0.88	0.92	0.87	0.90	0.89	0.83	0.86	0.88	0.82	0.85	0.87	0.81	0.84
Service	0.95	0.83	0.89	0.90	0.74	0.81	0.94	0.80	0.86	0.93	0.79	0.86	0.92	0.77	0.84	0.87	0.70	0.78
avg.	**0.94**	**0.90**	**0.92**	**0.92**	**0.88**	**0.89**	**0.95**	**0.90**	**0.93**	**0.94**	**0.89**	**0.91**	**0.94**	**0.87**	**0.90**	**0.92**	**0.86**	**0.89**

Table 4 Precision, recall and f1-scores obtained with the GAN-classifiers at different epochs (training dataset enriched with synthetic samples generated by the WGAN). Average values are in bold

	1,000 epochs ($GAN\text{-}classifier_1$)			10,000 epochs ($GAN\text{-}classifier_2$)			25,000 epochs ($GAN\text{-}classifier_3$)			50,000 epochs ($GAN\text{-}classifier_4$)			75,000 epochs ($GAN\text{-}classifier_5$)			100,000 epochs ($GAN\text{-}classifier_6$)		
Class	prec.	recall	f1 score	prec.	recall	f1 score	prec.	recall	f1 score	prec.	recall	f1 score	prec.	recall	f1 score	prec.	recall	f1 score
Exchange	0.96	0.98	0.97	0.96	0.98	0.97	0.96	0.98	0.97	0.96	0.98	0.97	0.96	0.98	0.97	0.96	0.98	0.97
Gambling	0.95	0.91	0.93	0.94	0.91	0.93	0.94	0.90	0.92	0.95	0.92	0.94	0.95	0.91	0.93	0.95	0.92	0.93
Market	0.99	0.97	0.98	0.99	0.97	0.98	0.99	0.97	0.98	0.99	0.98	0.98	0.99	0.98	0.98	0.99	0.98	0.98
Pool	0.86	0.76	0.81	0.85	0.76	0.80	0.80	0.78	0.79	0.81	0.78	0.80	0.79	0.79	0.79	0.79	0.80	0.79
Mixer	0.86	0.81	0.84	0.89	0.83	0.86	0.87	0.81	0.84	0.89	0.84	0.87	0.89	0.83	0.86	0.90	0.84	0.87
Service	0.87	0.74	0.80	0.90	0.74	0.81	0.83	0.72	0.77	0.90	0.78	0.83	0.89	0.78	0.83	0.90	0.76	0.83
avg.	**0.92**	**0.86**	**0.89**	**0.92**	**0.87**	**0.89**	**0.90**	**0.86**	**0.88**	**0.92**	**0.88**	**0.90**	**0.91**	**0.88**	**0.90**	**0.92**	**0.88**	**0.90**

Table 5 Precision, recall and f1-scores obtained with the GAN-classifiers at different epochs (training dataset enriched with synthetic samples generated by the unrolled GAN). Average values are in bold

	1,000 epochs ($GAN\text{-}classifier_1$)			10,000 epochs ($GAN\text{-}classifier_2$)			25,000 epochs ($GAN\text{-}classifier_3$)			50,000 epochs ($GAN\text{-}classifier_4$)			75,000 epochs ($GAN\text{-}classifier_5$)			100,000 epochs ($GAN\text{-}classifier_6$)		
Class	prec.	recall	f1 score	prec.	recall	f1 score	prec.	recall	f1 score	prec.	recall	f1 score	prec.	recall	f1 score	prec.	recall	f1 score
Exchange	0.96	0.98	0.97	0.96	0.98	0.97	0.96	0.98	0.97	0.96	0.98	0.97	0.95	0.98	0.97	0.95	0.98	0.97
Gambling	0.95	0.90	0.93	0.95	0.90	0.92	0.95	0.91	0.93	0.95	0.90	0.93	0.94	0.89	0.92	0.95	0.90	0.92
Market	0.99	0.97	0.98	0.99	0.97	0.98	0.99	0.97	0.98	0.99	0.97	0.98	0.99	0.97	0.98	0.99	0.97	0.98
Pool	0.89	0.78	0.83	0.93	0.80	0.86	0.93	0.80	0.86	0.91	0.78	0.84	0.91	0.78	0.84	0.91	0.77	0.84
Mixer	0.91	0.85	0.88	0.94	0.87	0.90	0.91	0.85	0.88	0.89	0.82	0.86	0.90	0.83	0.86	0.90	0.84	0.87
Service	0.92	0.77	0.84	0.88	0.72	0.79	0.92	0.76	0.83	0.92	0.77	0.83	0.89	0.71	0.79	0.89	0.71	0.79
avg.	**0.94**	**0.88**	**0.90**	**0.94**	**0.87**	**0.90**	**0.94**	**0.88**	**0.91**	**0.94**	**0.87**	**0.90**	**0.93**	**0.86**	**0.89**	**0.93**	**0.86**	**0.90**

Table 6 Overall accuracy, MCC and AUC of each GAN-classifier at different epochs separated per architecture

	Vanilla GAN			WGAN			unrolled GAN		
# epochs	score %	MCC	AUC	score %	MCC	AUC	score %	MCC	AUC
1,000	96.85	0.94	0.99	95.74	0.92	0.98	95.86	0.93	0.99
10,000	95.93	0.93	0.99	95.66	0.92	0.99	95.69	0.92	0.99
25,000	96.60	0.94	0.99	95.23	0.91	0.98	95.99	0.93	0.99
50,000	96.26	0.93	0.99	96.12	0.93	0.99	95.74	0.92	0.99
75,000	95.93	0.93	0.99	95.98	0.93	0.99	95.23	0.92	0.99
100,000	95.23	0.91	0.98	96.03	0.93	0.99	95.53	0.92	0.99

to any new knowledge ("low-valuable" samples). The two most promising configurations obtained with the Vanilla GAN, GAN-classifier$_1$ and GAN-classifier$_3$, were analysed in-depth in the third experiment.

5.3 Randomness of GANs

For the next experiment, GAN-classifier$_1$ and GAN-classifier$_3$ (trained for 1,000 and 25,000 epochs, respectively) were run five more times and tested. In particular, 5 new hybrid datasets were generated using 5 new generators. These new hybrid datasets were then used to train as many address classifiers (GAN-classifiers), each one tested again using the testing dataset.

In Table 7, the values of global accuracy, precision, recall and f1-scores as well as the averages of the metrics over the 5 repetitions and their standard deviations are shown for each new implementation. The best results were again obtained with the model based on the data generated from GANs trained for 1,000 epochs (GAN-classifier$_1$); its metrics showed good repeatability with 96.12% as the lowest value of accuracy and 96.96% as the greatest, with a global standard deviation of 0.282%. Slightly more unstable were the values obtained with models using data generated from GANs trained for 25,000 epochs (GAN-classifier$_3$). In this case, accuracy went down to 95.30% and up to 96.60%, with a global standard deviation of 0.472%.

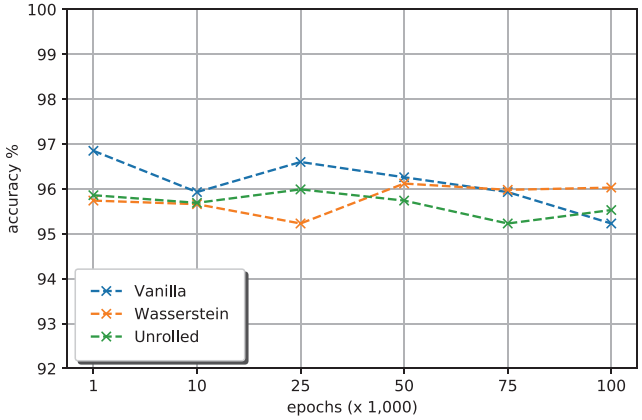


Fig. 11 Comparison of overall accuracy values obtained with the GAN-classifiers at different epochs

Figure 12 shows a comparison of f1-scores averaged over the 5 repetitions for each entity class. Both GAN-classifier$_1$ and GAN-classifier$_3$, presented very limited variability related to Exchange, Gambling and Market classification. Furthermore, GAN-classifier$_1$ presented high repeatability (limited variance) for the other three classes, while GAN-classifier$_3$ showed higher variability related to the Pool, Mixer and Service classes.

5.4 GANs for resampling vs. classical resampling methods

In the fourth and last experiment, the best solutions from prior experiments were compared to several common resampling techniques introduced in Section 2.3.

Table 8 indicates the per-class sample population after applying each additional resampling technique. Techniques such as RUS remove samples in order to reach the sample size present in the minority class, and techniques like ROS, SMOTE and ADASYN tend to create new samples in order to match the majority class. Interestingly, the Tomek links strategy did not change dramatically class distribution, even though it removes unwanted overlap between classes.

In Table 9, the values of overall accuracy, precision, recall, f1-score, MCC and AUC are reported, which were obtained by using the same testing dataset over the baseline model and over Bitcoin entity classifiers, in which the class imbalance problem was addressed by applying common resampling strategies and the novel GAN-based strategy introduced in this paper. The results shown in Table 9 can be compared with the baseline results and can be interpreted by dividing them into three categories: strategies that negatively affect classification results, strategies that do not seem to affect results and strategies that generate improvements.

The RUS technique falls into the first category as, in fact, this under-sampling algorithm removed important

Table 7 Overall accuracy, precision, recall and f1-scores obtained with GAN-classifier$_1$ and GAN-classifier$_3$ in 5 implementations

	1,000 (GAN-classifier$_1$)				25,000 (GAN-classifier$_3$)			
Repetition #	score %	prec.	recall	f1 score	score %	prec.	recall	f1 score
1	96.12	0.93	0.88	0.90	96.34	0.94	0.89	0.91
2	96.96	0.94	0.90	0.92	96.55	0.95	0.90	0.92
3	96.71	0.93	0.90	0.91	95.30	0.92	0.86	0.89
4	96.78	0.95	0.91	0.93	96.17	0.94	0.89	0.91
5	96.62	0.93	0.90	0.92	96.60	0.95	0.90	0.93
avg.	**96.64**	**0.94**	**0.90**	**0.92**	**96.19**	**0.94**	**0.89**	**0.91**
std.	**0.282**	**0.007**	**0.010**	**0.075**	**0.472**	**0.012**	**0.014**	**0.013**

In each one, the classifiers were trained with different hybrid datasets (newly generated using the best Vanilla GAN configuration). Average and standard values are in bold

information and generated a quality loss in the classification - reflected in a worsening of all the considered metrics. Accuracy decreased by more than 10% and the f1-score was 0.03 points below the baseline metric.

Tomek links and ADASYN were included in the second category. In particular, the under-sampling technique did not change strongly the initial dataset, and thus its evaluation metrics matched the baseline values. Although the ADASYN strategy generated improvements, we consider them to be insignificant here as the accuracy was just 0.12% and the f1-score was only 0.01 points above the baseline values.

ROS, SMOTE and GANs strategies belonged to the third category, as these techniques generated visible improvements in the results compared to values obtained by using the imbalanced Bitcoin dataset. In terms of overall accuracy, the best solution was obtained by addressing the imbalance problem using the GAN approach introduced in this paper, and in particular using GAN-classifier$_1$ i.e. a configuration of Vanilla GANs trained for 1,000 epochs. Our strategy showed an accuracy value of 96.64%; 0.07% above the second-best accuracy value achieved by the ROS strategy, and 1.97% above the baseline model. ROS and GAN strategies shared the same values of MCC and AUC (0.94 and 0.99, respectively). However, in terms of precision, recall and f1-score the ROS technique generated slightly better results, reaching an f1-score of 0.94 versus 0.92 obtained with the GAN strategy.

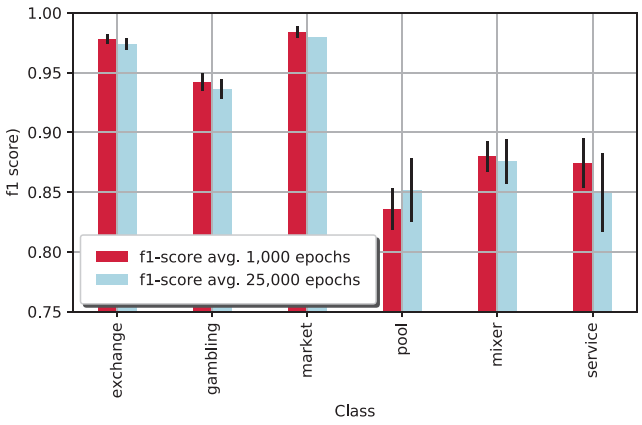


Fig. 12 Comparison of GAN-classifier$_1$ and GAN-classifier$_3$ f1-scores averaged over the 5 repetitions and standard deviations for each entity class

Furthermore, Table 9 shows that GAN-classifier$_3$ generated better results than the majority of the presented resampling techniques as well, in fact only GAN-classifier$_1$ and ROS performed better.

In Fig. 13, improvements and breakdowns in terms of precision are shown, and it is evident that the model trained with data enriched by our GAN strategy, together with ROS, were the best solutions. It is to be noted that the GAN strategy showed a decrease of precision regarding the Pool class (0.04 below baseline), while the model trained based on ROS data did not present such problems (0.04 above baseline). Yet, for the Gambling class, it was the GAN strategy that generated improvements (0.03 above the baseline), while the model trained with ROS data did not achieve this precision. Both techniques did not generate precision improvement for the Market class, but they generated similar improvements with respect to other classes. For example, in the Service class, a value of 0.1 points above the baseline model was obtained.

Figure 14 shows the breakdown of scores achieved per class in terms of recall. The best results were obtained using the SMOTE strategy. This strategy generated improvements in almost all classes. For the Service class, for example, the recall was 0.28 points above the baseline score; only in the Exchange class a lower score was registered (-0.03). The RUS trend was aligned with the theory of under-sampling techniques - removing random samples from the majority classes in order to balance the dataset, recall scores increased for underrepresented classes, while they were dramatically decreased for Exchange, Gambling and Market classes. Our GAN strategy improved recall values in all

Table 8 Populations in resampled datasets using 5 different resampling strategies

	RUS	TL	ROS	SMOTE	ADASYN
Class	Train size	Train size	Train size	Train size	Train size
Exchange	43,265	4,967,951	4,972,019	4,972,019	4,972,019
Gambling	43,265	1,524,202	4,972,019	4,972,019	4,975,061
Market	43,265	1,173,795	4,972,019	4,972,019	4,971,980
Pool	43,265	43,265	4,972,019	4,972,019	4,971,821
Mixer	43,265	236,155	4,972,019	4,972,019	4,972,000
Service	43,265	125,019	4,972,019	4,972,019	4,971,911

classes, except for the Market class. Nevertheless, these improvements were limited to a range between 0.01 and 0.14 points above baseline and were larger in ROS and SMOTE strategies.

In terms of f1-score (harmonic mean of precision and recall), the ROS implementation generated the highest improvements especially regarding underrepresented classes, with 0.09, 0.06 and 0.17 points above baseline values for Mining Pool, Mixer and Service classes, respectively (Fig. 15). Our GAN strategy followed the trend of the ROS strategy for Exchange, Gambling, Market and Mixer classes, however, for Pool and Service classes, f1-scores were only 0.02 and 0.13 points above baseline.

To investigate practical aspects regarding the implementation of different resampling techniques, Fig. 16 shows the computational cost of implementing the different techniques in terms of execution time. For state-of-the-art techniques, which do not require a training process, the time in seconds (s) they needed to create a more balanced training dataset via resampling is indicated. For GAN resampling, the time for training each GAN is now given in terms of seconds (s) for allowing a comparison. Computational costs were computed by performing resampling of the same input dataset on a server with 64GB of RAM and 16 vCPUs of 2.20 GHz.

As shown in Fig. 16, RUS and ROS techniques were the fastest strategies; they performed resampling in 8 and 17s, respectively. Meanwhile, the SMOTE algorithm needed 817s. The TL and ADASYN techniques were more costly, requiring hours for resampling the initial dataset (respectively 6,291 and 14,058s). The GAN strategies reached 1,000 epochs in a range of 409s to 521s; 25,000 epochs were reached in a range of 10,232s to 13,060s. This computational test confirmed the linear dependence of GAN training time - during the training process the execution time required to reach 25,000 epochs was about 25 times the one required to reach 1,000 epochs.

5.5 Discussion

In this study, an approach for using adversarial learning (GANs) to tackle class imbalance in Bitcoin data was introduced with the ultimate goal of decreasing Bitcoin entity anonymity. Our approach showed promising results that we can summarize as follows:

- All GAN architectures generated improvements in Bitcoin entity classification;
- The classifier trained with the Vanilla GAN-generated synthetic data yielded better results than classifiers trained with WGANs and unrolled GANs;
- Vanilla GAN implementations performed better using shorter training epochs;
- Vanilla GAN implementations using short training epochs achieved high repeatability;
- Vanilla GAN classifiers generated overall higher accuracy, precision, recall and f1-score than the baseline model (based on imbalanced data);

Table 9 Classification metrics obtained by models trained with datasets resampled with different strategies. The best value of each metric is highlighted

	Score %	prec.	recall	f1 score	MCC	AUC
Baseline	94.67	0.91	0.84	0.87	0.90	0.98
RUS	83.10	0.88	0.83	0.84	0.74	0.97
TL	94.67	0.91	0.84	0.87	0.90	0.98
ROS	96.57	**0.95**	**0.93**	**0.94**	**0.94**	0.99
SMOTE	95.70	0.85	0.96	0.90	0.93	0.99
ADASYN	94.79	0.90	0.87	0.88	0.91	0.99
GAN-classifier$_1$	**96.64**	0.94	0.90	0.92	**0.94**	0.99
GAN-classifier$_3$	96.20	0.94	0.89	0.91	0.93	0.99

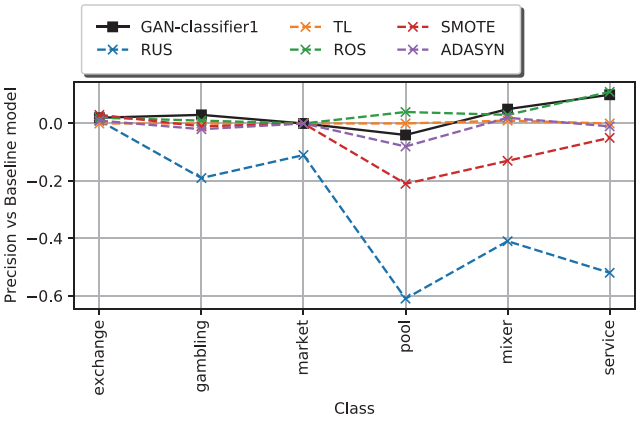


Fig. 13 Improvements compared to baseline and breakdown in terms of precision

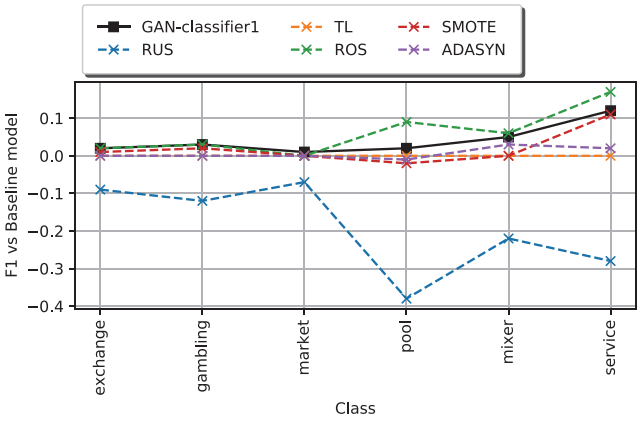


Fig. 15 Improvements compared to baseline and breakdown in terms of f1-score

- The best GAN-based classifiers were in the top 3 techniques for classification improvements in terms of all considered metrics (overall accuracy, precision, recall, f1-score, MCC and AUC) compared to other resampling techniques (together with ROS);
- Except for the two random resampling techniques RUS and ROS, the GAN technique was the most efficient strategy in terms of single task computational costs.

Limitations and strengths

The model trained with data generated by the Vanilla GAN architecture reached higher values of accuracy than models trained with data generated from WGANs and unrolled GANs. However, as shown in Tables 3, 4 and 5, none of the classifiers generated sufficient improvements for the Pool and Service classes (f1-scores below 0.90), regardless of the considered GAN architectures and the number of epochs. This effect motivates two hypotheses regarding possible limitations of the dataset and limitations of the GAN approach. On the one hand, the issue could be related to the distribution of those affected classes in the training dataset. In this case, after splitting into training and testing datasets, the training set may not have enough sample variety for describing all possible behaviours, which causes a loss of quality during the GAN training. On the other hand, it could happen that - although presenting a sparse distribution - affected classes are characterized by several high-density regions in the feature space. This phenomenon may affect GAN training such that only behaviours in these high-density regions are learnt.

The best values were obtained using a Vanilla GAN architecture by training this network with a lower number of epochs (1,000). In fact, too long training of this architecture (using a high number of epochs) caused a decrease of classification accuracy, probably due to known downside effects of GANs as discussed in Section 2.2 (the mode collapse effect - overfitting).

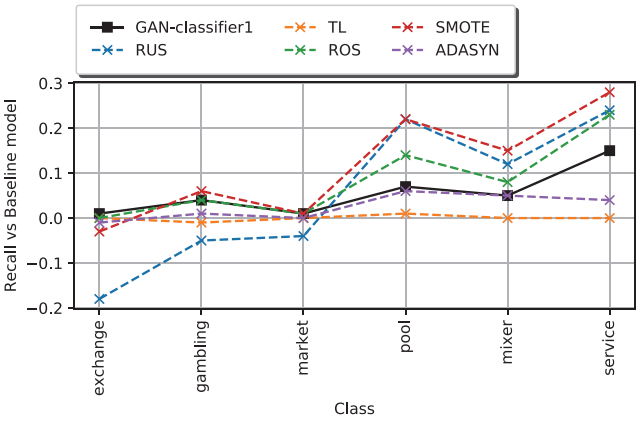


Fig. 14 Improvements compared to baseline and breakdown in terms of recall

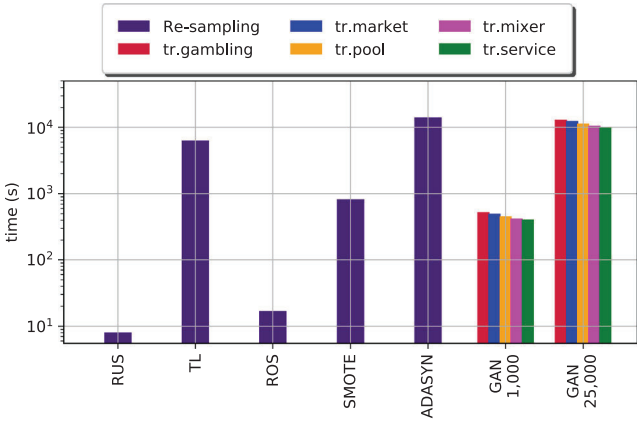


Fig. 16 Execution times of different resampling strategies for the same input dataset. For GAN strategies, the training time is indicated

This behaviour is highlighted in Fig. 11, which shows the downward trend of obtained accuracy with increased training time related to the Vanilla GAN. The same Figure highlights how the WGAN was affected in a contrary way. Results showed that more training time (more epochs) was required for training the WGANs and for obtaining more valuable synthetic samples. However, within the range of epochs considered in the experiments, the accuracy of the model trained with the WGAN architecture was lower than the accuracy achieved by the classifier trained with the Vanilla GAN.

The two best GAN configurations we found, the Vanilla GAN trained with 1,000 and 25,000 epochs (GAN-classifier$_1$ and GAN-classifier$_3$), were tested another 5 times changing the composition of the training dataset used for training each class GAN, and they showed that the most stable (least varying) classifier with highest accuracy value was obtained by training GANs in a short period (1,000 epochs).

Comparison to other resampling techniques

The fourth experiment demonstrated once more that the class imbalance problem is an important problem in the Bitcoin entity classification task and that it strongly affects the quality of the classifiers. All used over-sampling techniques (ROS, SMOTE and ADASYN) generated improvements in terms of recall, f1-score, MCC and AUC, while under-sampling techniques (RUS and TL) performed poorly (Table 9).

Comparing the GAN-based strategy with 5 frequently used resampling techniques, our approach represented the best solution in terms of accuracy with 96.64%; 1.97% more than the baseline accuracy. The GAN-classifier$_1$ together with ROS reached also the highest overall MCC score of 0.94. However, in terms of overall precision, recall and f1-score the ROS strategy outperformed the GAN strategy slightly.

These results, together with the results obtained in the third experiment (repeatability of the results) definitively invalidate our first hypothesis related to the limitation of the training dataset distribution. In fact, they show that simple replication of the training samples (ROS algorithms) generates good improvements in the Pool and Service classes as well (Fig. 15). However, the small improvements generated with the GAN resampled datasets confirm our second hypothesis, that GANs learn partial information of Pool and Service classes only, likely due to the sparse distribution of their behaviours and the presence of high-density regions in the feature space. This assumption is confirmed by analysing Table 9 where GAN-classifiers show high values of precision (i.e. low false positives) and low values of recall (i.e. high false negatives).

Finally, since GANs technologies help to generate samples of specific entities with high variability, they are less prone to overfitting than the ROS technique. Further, following the results obtained, this sampling variability does not strongly affect the classification performance as it is the case of the other techniques generating higher variability like SMOTE and ADASYN. This demonstrates that not only in image applications [82], but also in Bitcoin applications, GANs are able to learn the data distribution and generate new samples starting from a very limited class representation (very high imbalance ratio)

Comparison of execution times

In terms of execution cost, the two simple resampling techniques (RUS and ROS) beat all the others in terms of computation speed (less than 20s) by a lot. However, comparing the remaining resampling techniques, the GAN strategy presents a high cumulative training time. Nevertheless, using a parallel training process, the GAN technique trained in 1,000 epochs became the next best solution after RUS and ROS, as shown by its fast-training time (average $\sim$ 460s for each implementation). The training process could be parallelized by training the 5 GANs at the same time using different threads. In fact, a single GAN task was 1.77 times faster than the whole SMOTE process, and respectively 13.7 and 30.5 faster than the TL and ADASYN algorithms.

Comparison with previous work

In Table 10, our GAN-based classification results are compared with previous studies that directly apply Artificial Neural Network (ANN), Random Forest (RF) [83], Gradient Boosting Classifiers (GBC), GBC combined with SMOTE [7], Logistic Regression (LR) and two implementations of Light Gradient Boosting Machine (LGBM) [6, 36] in terms of overall and per class f1-score. Our approach is the best in terms of overall f1-score when using ROS (0.94) and shares second place with the Light Gradient Boosting Machine (LGBM) when using the GAN-based method. In particular, our GAN-based approach represents the best solution for detecting Exchange (0.98) and Market (0.94) entities by a difference of several points compared to previously published models. Further, it is in second place for detecting Gambling (0.95) and Service (0.89) classes (slightly outperformed by our ROS implementation for the latter), while it has problems detecting Pool entities. More specifically, the four classes in which our GAN approach

Table 10 Comparison of per-class and overall f1-score achieved by comparable state-of-the-art works. The best value, for each class, is highlighted

	f1-score								
	[83]	[7]	[6]	[36]	Proposed method				
Class	RF	ANN	GBC	GBC (SMOTE)	LR	LGBM	LGBM	ROS	GAN classifier$_1$
Exchange	0.78	0.56	0.86	0.84	0.91	0.92	0.89	**0.98**	**0.98**
Gambling	0.77	0.48	0.78	0.78	0.82	**0.97**	0.83	0.94	0.95
Market	-	-	0	0	-	-	0.78	0.98	**0.99**
Pool	0.86	0.65	0.90	0.86	0.67	0.67	0.83	**0.91**	0.82
Mixer	0.82	0.45	0.33	0.97	-	-	**0.98**	0.89	0.88
Service/Other*	-	-	0.22*	0.08*	0.87	0.88	-	**0.92**	0.89
Overall	-	-	0.75	0.76	0.87	0.92	0.87	**0.94**	0.92

performs better are also the ones that are interesting from an investigation point of view since those four, together with Mixers, represent the entities more prone to be involved in illicit transactions such as money laundering (Section 2.1). It is to be noted that our ROS and GAN-based approaches together outperform all other models for the majority of classes except LGBM for the Gambling and Mixer classes. Further, our implementations generate overall precision, recall and f1-score scores that are aligned with the results presented in [70] (between 0.90 and 0.94), however, we achieve better overall accuracy ($\sim$ 5%), and better recall values per class, especially for Exchange, Mining, Mixer and Gambling classes.

Finally, comparing our GAN-based approach with work presented in [78], our metrics exceed their results (with respect to WCGAN, DRAGAN and SMOTE) in terms of AUC ($\sim$ 0.07) and recall ($\sim$ 0.59), even though precision decreases slightly ($\sim$ 0.05).

6 Conclusions

Bitcoin entity classification is an important task in cryptocurrency network analysis for decreasing entity anonymity, for detecting classes related to abnormal or illicit activities and for increasing the network's resilience to cyber-attacks (detecting the more vulnerable classes). Typically, this task is performed by applying supervised machine learning algorithms whose results are strongly conditioned by ground truth Bitcoin datasets. Yet, these labelled input datasets usually do not contain a homogeneous population of the various Bitcoin entity classes, yielding a class imbalance problem, which results in poor classification performance.

Current blockchain studies are transferring a variety of technologies that have already shown good results in other domains to the Bitcoin blockchain domain. Following this trend, in this work, we introduced the application of Generative Adversarial Networks (GANs) to address the Bitcoin imbalance problem with the aim to evaluate how this synthetic information can improve the entity classification task. GANs have mainly been used in the image or video processing domain and specifically for addressing imbalance problems. In this work, we investigated different types of GAN architectures and how we can apply them to the Bitcoin domain in order to address the imbalance problem and improve the final multi-class classification. Experiments were performed to check the repeatability of GAN-based resampling and to compare our approach to other common resampling techniques.

Our GAN-based classification approach generally obtained promising results, achieving the best results in terms of accuracy and was among the best in terms of precision, recall and f1-score compared to other resampling techniques. Furthermore, the presented methods outperformed previously published models with regard to Exchange, Market, Pool and Service entity classification.

Along the way, we detailed and highlighted here GAN-specific characteristics such as the influence of training epochs, which need to be taken into consideration when applying GANs to Bitcoin-related data. Finally, we detailed the limitations of this approach when the training dataset is composed of dense areas in the features space. In these cases, such distributions seem to force GANs to learn certain behaviours and forget others.

This study represents a first step towards the usage of GAN technologies with Bitcoin behavioural data. Motivated by our promising results, future work could be directed towards testing other optimization functions (like Adagrad [84] or Adadelta [85]), evaluating changes in G and D networks. In this way, it will be possible to evaluate if with the help of a more robust optimizer the GANs will learn to generate more valuable samples (in terms of classification improvements). Further, it will be interesting to implement multi-GAN solutions for each class, i.e. training more than one GAN for each class in order to increase the

knowledge about the training dataset and evaluate their effects in generating information. Based on the promising results presented in this paper, future studies could focus on analysing alternative data augmentation techniques such as Mixup [86] for Bitcoin entity classification. Finally, it could be interesting to apply and validate the introduced approach to address class imbalance problems using other machine learning models different from Random Forest, for example by including Neural Networks or tensor-based classifiers [87]. Overall, our study has shown that by applying carefully selected resampling techniques the Bitcoin class imbalance problem can indeed be tackled, leading to very good classification results across a broad range of critical (and often under-represented) Bitcoin entity classes, which may ultimately impact positively on Bitcoin de-anonymization and the detection of illicit activities within the network.

Acknowledgements This work has been partially supported by the Spanish Centre for the Development of Industrial Technology (CDTI) under the project ÉGIDA (CER20191012) - RED DE EXCELENCIA EN TECNOLOGIAS DE SEGURIDAD Y PRIVACIDAD.

Author Contributions F. Zola: Conceptualization of this study, Related Work, Methodology, Results, Discussion, Writing - Original draft preparation
L. Segurola-Gil: Related Work, Data curation,
J.L. Bruse: Discussion, Writing - Review and editing
M. Galar: Discussion, Writing - Review and editing
R. Orduna-Urrutia: Conceptualization of this study, Supervision

Funding Not applicable

Declarations

Availability of Data and Material (Data Transparency) Not applicable

Code Availability Not applicable

Competing interests (check journal-specific guidelines for which heading to use)

Conflict of Interests Not applicable

References

1. Nakamoto S (2019) Bitcoin: A peer-to-peer electronic cash system. Technical Report, Manubot
2. Foley S, Karlsen JR, Putniņš TJ (2019) Sex, drugs, and bitcoin: How much illegal activity is financed through cryptocurrencies? Rev Financ Stud 32(5):1798–1853
3. Marella V, Upreti B, Merikivi J, Tuunainen VK (2020) Understanding the creation of trust in cryptocurrencies: the case of bitcoin. Electron Mark:1–13
4. Saez M (2020) Blockchain-enabled platforms: Challenges and recommendations. Int J Interact Multimed Artif Intell 6(3)
5. Zola F, Bruse JL, Eguimendia M, Galar M, Orduna Urrutia R (2019) Bitcoin and cybersecurity: temporal dissection of blockchain data to unveil changes in entity behavioral patterns. Appl Sci 9(23):5003
6. Jourdan M, Blandin S, Wynter L, Deshpande P (2018) Characterizing entities in the bitcoin blockchain. In: IEEE International Conference on Data Mining Workshops (ICDMW). IEEE, pp 55–62
7. Harlev MA, Sun Yin H, Langenheldt KC, Mukkamala R, Vatrapu R (2018) 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
8. Fernández A, García S, Galar M, Prati RC, Krawczyk B, Herrera F (2018) Learning from imbalanced data sets. Springer
9. Monamo PM, Marivate V, Twala B (2016) A multifaceted approach to bitcoin fraud detection: Global and local outliers. In: 15th IEEE International Conference on Machine Learning and Applications (ICMLA). IEEE, pp 188–194
10. Zheng W, Zhao H (2020) Cost-sensitive hierarchical classification for imbalance classes. Appl Intell 50(8):2328–2338
11. Fernández A, García S, Galar M, Prati RC, Krawczyk B, Herrera F (2018) Algorithm-level approaches. In: Learning from imbalanced data sets. Springer, pp 123–146
12. Wang S, Liu W, Wu J, Cao L, Meng Q, Kennedy PJ (2016) Training deep neural networks on imbalanced data sets. In: 2016 international joint conference on neural networks (IJCNN). IEEE, pp 4368–4374
13. Manju N, Harish BS, Nagadarshan N (2020) Multilayer feedforward neural network for internet traffic classification. Int J Interact Multim Artif Intell 6(1):117–122
14. Alotaibi A (2020) Deep generative adversarial networks for image-to-image translation: a review. Symmetry 12(10):1705
15. Brock A, Donahue J, Simonyan K (2018) Large scale gan training for high fidelity natural image synthesis. In: International conference on learning representations
16. Vondrick C, Pirsiavash H, Torralba A (2016) Generating videos with scene dynamics. Adv Neural Inf Proces Syst 29
17. Bowles C, Chen L, Guerrero R, Bentley P, Gunn R, Hammers A, Dickie DA, Hernández MV, Wardlaw J, Rueckert D (2018) Gan augmentation: Augmenting training data using generative adversarial networks. arXiv:1810.10863
18. Abusitta A, Aïmeur E, Abdel Wahab O (2020) Generative adversarial networks for mitigating biases in machine learning systems. In: ECAI 2020. IOS Press, pp 937–944
19. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: Advances in neural information processing systems, pp 2672–2680
20. Wang K, Gou C, Duan Y, Lin Y, Zheng X, Wang F-Y (2017) Generative adversarial networks: introduction and outlook. IEEE/CAA J Autom Sin 4(4):588–598

21. Shamsolmoali P, Zareapoor M, Wang R, Jain DK, Yang J (2019) G-ganisr: Gradual generative adversarial network for image super resolution. Neurocomputing 366:140–153
22. Ledig C, Theis L, Huszar F, Caballero J, Cunningham A, Acosta A, Aitken A, Tejani A, Totz J, Wang Z et al (2017) Photo-realistic single image super-resolution using a generative adversarial network. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4681–4690
23. Alqahtani H, Kavakli-Thorne M, Kumar G (2019) Applications of generative adversarial networks (gans): an updated review. Arch Comput Methods Eng:1–28
24. Kethineni S, Cao Y (2019) The rise in popularity of cryptocurrency and associated criminal activity. Int Crim Justice Rev:1057567719827051
25. Hu M, Chen J, Gan W, Chen C-M (2021) A jumping mining attack and solution. Appl Intell 51(3):1367–1378
26. Fanusie Y, Robinson T (2018) Bitcoin laundering: an analysis of illicit flows into digital currency services. Center on Sanctions & Illicit Finance memorandum
27. Sun X, Yang T, Hu B (2021) Lstm-tc: Bitcoin coin mixing detection method with a high recall. Appl Intell:1–14
28. Conti M, Kumar ES, Lal C, Ruj S (2018) A survey on security and privacy issues of bitcoin. IEEE Commun Surv Tutorials 20(4):3416–3452
29. Zola F, Eguimendia M, Bruse JL, Urrutia RO (2019) Cascading machine learning to attack bitcoin anonymity. In: IEEE International conference on blockchain (Blockchain). IEEE, pp 10–17
30. Yin HS, Vatrapu R (2017) A first estimation of the proportion of cybercriminal entities in the bitcoin ecosystem using supervised machine learning. In: IEEE International conference on big data (Big Data). IEEE, pp 3690–3699
31. Hu M, Chen J, Gan W, Chen C-M (2020) A jumping mining attack and solution. Appl Intell 51(3):1367–1378
32. Kim S, Kim B, Kim HJ (2018) Intrusion detection and mitigation system using blockchain analysis for bitcoin exchange. In: Proceedings of the international conference on cloud computing and internet of things, pp 40–44
33. Zhang Y, Wang J, Luo J (2020) Heuristic-based address clustering in bitcoin. IEEE Access 8:210582–210591
34. Paquet-Clouston M, Haslhofer B, Dupont B (2019) Ransomware payments in the bitcoin ecosystem. J Cybersecur 5(1):tyz003
35. Haslhofer B, Karl R, Filtz E (2016) O bitcoin where art thou? insight into large-scale transaction graphs. In: SEMANTiCS (Posters, Demos, SuCCESS)
36. Lin Y-J, Wu P-W, Hsu C-H, Tu I-P, Liao S (2019) An evaluation of bitcoin address classification based on transaction history summarization. In: IEEE International Conference on Blockchain and Cryptocurrency (ICBC). IEEE, pp 302–310
37. Liao K, Zhao Z, Doupé A, Ahn G-J (2016) Behind closed doors: measurement and analysis of cryptolocker ransoms in bitcoin. In: APWG Symposium on Electronic Crime Research (eCrime). IEEE, pp 1–13
38. Farnia F, Ozdaglar A (2020) Do gans always have nash equilibria? In: International conference on machine learning. PMLR, pp 3029–3039
39. Yuan W, Hu F, Lu L (2021) A new non-adaptive optimization method: stochastic gradient descent with momentum and difference. Appl Intell:1–15
40. Sun R, Fang T, Schwing A (2020) Towards a better global loss landscape of gans. Adv Neural Inf Process Syst 33
41. Goodfellow I (2016) Nips 2016 tutorial: Generative adversarial networks. arXiv:1701.00160
42. Dai Y, Wang S, Chen X, Xu C, Guo W (2020) Generative adversarial networks based on wasserstein distance for knowledge graph embeddings. Knowl-Based Syst 190:105165
43. Martin A, Lon B (2017) Towards principled methods for training generative adversarial networks. In: NIPS Workshop on adversarial training. In review for ICLR, vol 2016
44. Arjovsky M, Chintala S, Bottou L (2017) Wasserstein generative adversarial networks. In: Precup D, Teh YW (eds) Proceedings of the 34th international conference on machine learning, vol 70. PMLR, pp 214–223
45. Nagarajan V, Kolter JZ (2017) Gradient descent gan optimization is locally stable. In: Advances in neural information processing systems, pp 5585–5595
46. Metz L, Poole B, Pfau D, Sohl-Dickstein J (2017) Unrolled generative adversarial networks. In: 5th International conference on learning representations, conference track proceedings. OpenReview.net
47. Sahu S, Gupta R, Espy-Wilson C (2018) On enhancing speech emotion recognition using generative adversarial networks. In: INTERSPEECH
48. Yi X, Walia E, Babyn P (2019) Generative adversarial network in medical imaging: a review. Med Image Anal 58:101552
49. Ali-Gombe A, Elyan E (2019) Mfc-gan: class-imbalanced dataset classification using multiple fake class generative adversarial network. Neurocomputing 361:212–221
50. Dai X, Yuan X, Wei X (2021) Data augmentation for thermal infrared object detection with cascade pyramid generative adversarial network. Appl Intell:1–15
51. Liu Q-M, Jia R-S, Liu Y-B, Sun H-B, Yu J-Z, Sun H-M (2021) Infrared image super-resolution reconstruction by using generative adversarial network with an attention mechanism. Appl Intell 51(4):2018–2030
52. Zhang F, Ma Y, Yuan G, Zhang H, Ren J (2021) Multiview image generation for vehicle reidentification. Appl Intell:1–18
53. Zong X, Chen Z, Wang D (2021) Local-cyclegan: a general end-to-end network for visual enhancement in complex deep-water environment. Appl Intell 51(4):1947–1958
54. Chen Y, Zhang H, Liu L, Chen X, Zhang Q, Yang K, Xia R, Xie J (2021) Research on image inpainting algorithm of improved gan based on two-discriminations networks. Appl Intell 51(6):3460–3474
55. Chen S, Chen S, Guo Z, Zuo Y (2019) Low-resolution palmprint image denoising by generative adversarial networks. Neurocomputing 358:275–284
56. Li Y, Zhang Y, Yu K, Hu X (2021) Adversarial training with wasserstein distance for learning cross-lingual word embeddings. Appl Intell:1–13
57. Yang Z, Chen W, Wang F, Xu B (2018) Generative adversarial training for neural machine translation. Neurocomputing 321:146–155
58. Athanasiadis C, Hortal E, Asteriadis S (2019) Audio–visual domain adaptation using conditional semi-supervised generative adversarial networks. Neurocomputing
59. Merino T, Stillwell M, Steele M, Coplan M, Patton J, Stoyanov A, Deng L (2019) Expansion of cyber attack data from unbalanced datasets using generative adversarial networks. In: International conference on software engineering research, management and applications. Springer, pp 131–145
60. Yilmaz I, Masum R (2019) Expansion of cyber attack data from unbalanced datasets using generative techniques. arXiv:1912.04549
61. Mukhtar N, Batina L, Picek S, Kong Y (2021) Fake it till you make it: Data augmentation using generative adversarial networks for all the crypto you need on small devices
62. Wang P, Li S, Ye F, Wang Z, Zhang M (2020) Packetcgan: Exploratory study of class imbalance for encrypted traffic classification using cgan. In: ICC 2020-2020 IEEE International Conference on Communications (ICC). IEEE, pp 1–7

63. Haixiang G, Yijing L, Shang J, Mingyun G, Yuanyue H, Bing G (2017) Learning from class-imbalanced data: review of methods and applications. Expert Syst Appl 73:220–239
64. García V, Sánchez JS, Marqués AI, Florencia R, Rivera G (2019) Understanding the apparent superiority of over-sampling through an analysis of local information for class-imbalanced data. Expert Syst Appl:113026
65. Pereira RM, Costa YandreMG, Silla Jr C N (2020) Mltl: A multi-label approach for the tomek link undersampling algorithm. Neurocomputing 383:95–105
66. Fernandez A, Garcia S, Herrera F, Chawla NV (2018) Smote for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. J Artif Intell Res 61:863–905
67. Vo MT, Nguyen T, Vo HA, Le T (2021) Noise-adaptive synthetic oversampling technique. Appl Intell:1–10
68. Oussidi A, Elhassouny A (2018) Deep generative models: Survey. In: 2018 International Conference on Intelligent Systems and Computer Vision (ISCV). IEEE, pp 1–8
69. Xie Y, Zhang T (2018) Imbalanced learning for fault diagnosis problem of rotating machinery based on generative adversarial networks. In: 2018 37th Chinese Control Conference (CCC). IEEE, pp 6017–6022
70. Nerurkar P, Bhirud S, Patel D, Ludinard R, Busnel Y, Kumari S (2021) Supervised learning model for identifying illegal activities in bitcoin. Appl Intell 51(6):3824–3843
71. Bartoletti M, Pes B, Serusi S (2018) Data mining for detecting bitcoin ponzi schemes. In: Crypto Valley Conference on Blockchain Technology (CVCBT). IEEE, pp 75–84
72. Ranshous S, Joslyn CA, Kreyling S, Nowak K, Samatova NF, West CL, Winters S (2017) Exchange pattern mining in the bitcoin transaction directed hypergraph. In: International conference on financial cryptography and data security. Springer, pp 248–263
73. Liang J, Li L, Luan S, Gan L, Zeng D (2019) Bitcoin exchange addresses identification and its application in online drug trading regulation. In: 23rd Pacific Asia Conference on Information Systems: Secure ICT Platform for the 4th Industrial Revolution, PACIS 2019
74. Monamo MP (2018) Anomaly detection in the open financial markets: A case for the bitcoin network. University of Johannesburg, South Africa
75. Pfenninger M, Rikli S, Bigler DN (2021) Wasserstein gan: Deep generation applied on financial time series. Available at SSRN 3877960
76. Grilli L, Santoro D (April 2020) Generative Adversarial Network for Market Hourly Discrimination. In: 3RD International conference on mathematical and related sciences: current trends and developments proceedings book
77. Zola F, Bruse JL, Barrio XE, Galar M, Urrutia RO (2020) Generative adversarial networks for bitcoin data augmentation. In: 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS). IEEE, pp 136–143
78. Han J, Woo J, Hong JW-K (2020) Oversampling techniques for detecting bitcoin illegal transactions. In: 21st Asia-Pacific Network Operations and Management Symposium (APNOMS). IEEE, pp 330–333
79. Toyoda K, Ohtsuki T, Mathiopoulos PT (2018) Multi-class bitcoin-enabled service identification based on transaction history summarization. In: 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). IEEE, pp 1153–1160
80. Zou F, Shen L, Jie Z, Zhang W, Liu W (2019) A sufficient condition for convergences of adam and rmsprop. In: Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition, pp 11127–11135
81. Chicco D, Jurman G (2020) The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. BMC Genomics 21(1):1–13
82. Sampath V, Maurtua I, Martín JJA, Gutierrez A (2021) A survey on generative adversarial networks for imbalance problems in computer vision tasks. J Big Data 8(1):1–59
83. Lee C, Maharjan S, Ko K, Woo J, Hong JW-K (2020) Machine learning based bitcoin address classification. In: International conference on blockchain and trustworthy systems. Springer, pp 517–531
84. Lydia A, Francis S (2019) Adagrad–an optimizer for stochastic gradient descent. Int J Inf Comput Sci 6(5)
85. Dogo EM, Afolabi OJ, Nwulu NI, Twala B, Aigbavboa CO (2018) A comparative analysis of gradient descent-based optimization algorithms on convolutional neural networks. In: 2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS). IEEE, pp 92–99
86. Liang D, Yang F, Zhang T, Yang P (2018) Understanding mixup training methods. IEEE Access 6:58774–58783
87. Hu C, He S, Wang Y (2021) A classification method to detect faults in a rotating machinery based on kernelled support tensor machine and multilinear principal component analysis. Appl Intell 51(4):2609–2621

Francesco Zola obtained his Bachelors degree in Telecommunication Engineering at University of Cassino and Southern Lazio, Italy, in 2012 and his Master's degree in Computer Science in the same university, in 2015. He works as associate researcher and data scientist in Vicomtech in the department of Digital Security applying machine learning in cybersecurity projects. In 2019 he started his doctorate in collaboration with the Public University of Navarre (UPNA), focusing his works on graph analysis, machine learning, data augmentation and classification. Francesco is involved in several European and industrial projects related to cybersecurity, blockchain analysis and anomaly detection.

Lander Segurola-Gil is a mathematician who obtained his Bachelors degree in the University of País Vasco Euskal Herriko Unibertsitatea (UPV-EHU) and received a Masters degree in Mathematics and Applications from the University of Madrid (UAM). Since 2019, he works as a research assistant in Vicomtech in the department of Digital Security. His research interests are Naive Bayes networks, attack detection and machine learning applied to time series.

Dr. Jan L. Bruse graduated in Mechanical Engineering at RWTH Aachen University, Germany, in 2013 and received his doctorate in Biomedical Engineering from University College London, United Kingdom, in 2017. His PhD thesis combines Medical Image Analysis, Computational Simulation and Machine Learning for the development of Clinical Decision Support Systems. Jan has participated in several international and interdisciplinary research collaborations, has multiple peer-reviewed publications in journals and congresses indexed in the engineering and clinical sector and is reviewer for several international journals. Since September 2017, Jan is Research Engineer at Vicomtech, Spain, in the area of Data Intelligence for Energy and Industrial Processes where he develops Artificial Intelligence (AI) and Machine Learning platforms within the context of Industry 4.0 projects.

Dr. Mikel Galar received the MSc and PhD degrees in Computer Science in 2009 and 2012, both from the Public University of Navarre (UPNA), Spain. He is currently an associate professor at the Department of Statistics, Computer Science and Mathematics at the UPNA. He is the author of 35 published original articles in international journals and more than 50 contributions to conferences. He is also reviewer of more than 35 international journals. His research interests are machine learning, data mining, classification, fuzzy systems and big data. He is a member of the IEEE, the European Society for Fuzzy Logic and Technology (EUSFLAT) and the Spanish Association of Artificial Intelligence (AEPIA). He has received the extraordinary prize for his PhD thesis from the Public University of Navarre and the 2013 IEEE Transactions on Fuzzy System Outstanding Paper Award for the paper "A New Approach to Interval-Valued Choquet Integrals and the Problem of Ordering in Interval-Valued Fuzzy Set Applications" (bestowed in 2016).

Dr. Raul Orduna-Urrutia received his degree in Computer Engineering at the Faculty of Informatics of San Sebastian by the University of the Basque Country (UPV/EHU) and obtained a PhD degree in Computer Science and Artificial Intelligence at the School of Industrial and Telecommunications Engineering (ETSIIT) of the Public University of Navarre (UPNA). He is currently an associate professor at the Department of Statistics and Computer Science at the UPNA, and, at the same time, the Digital Security Director in Vicomtech. He has taken part or led projects related with ethical hacking, forensic analysis, malware analysis, access control and cryptography. He has participated in more than 6 successful funded projects, is the author of 6 published original articles in international journals. He is also a reviewer of Fuzzy Sets and Systems.

Affiliations

Francesco Zola[1,2] · Lander Segurola-Gil[1] · Jan L. Bruse[1] · Mikel Galar[2] · Raul Orduna-Urrutia[1]

Lander Segurola-Gil
lsegurola@vicomtech.org

Jan L. Bruse
jbruse@vicomtech.org

Mikel Galar
mikel.galar@unavarra.es

Raul Orduna-Urrutia
rorduna@vicomtech.org

[1] Digital Security, Vicomtech Foundation, Basque Research and Technology Alliance (BRTA), Paseo Mikeletegi, Donostia/San Sebastian, 20009, Spain

[2] Institute of Smart Cities, Public University of Navarre, Pamplona, 31006, Spain

4. *Cyber Threat Intelligence for Malware Classification in the Presence of Concept Drift*

The work related to this part is in:

- Zola F., Bruse J.L., Galar M. and Cavallaro L., *Cyber Threat Intelligence for Malware Classification in the Presence of Concept Drift*

- Status: Submitted

Cyber Threat Intelligence for Malware Classification in the Presence of Concept Drift

Francesco Zola*†
fzola@vicomtech.org
Vicomtech Fundation
Spain

Jan L. Bruse
jbruse@vicomtech.org
Vicomtech Fundation
Spain

Mikel Galar
mikel.galar@unavarra.es
Public University of Navarre
Spain

Lorenzo Cavallaro
l.cavallaro@ucl.ac.uk
University College London
UK

Abstract

In recent years, malware diversity and complexity have increased substantially so that machine learning (ML)-based approaches have been proposed to detect and classify the growing number of malware families. However, these approaches are most focused on achieving high classification performance scores in static scenarios, without taking into account a key feature of malware: it is constantly evolving. This leads to ML models being outdated and performing poorly after only a few months, leaving stakeholders exposed to potential security risks. With this work, we aim to highlight the issues that may arise when applying ML-based classification to malware data and propose a three-step approach to carry out an in-depth analysis favouring interpretation of the results for enhancing Cyber Threat Intelligence (CTI). First, we perform a multi-class classification using Control Flow Graph (CFG) data extracted from portable executables, followed by temporal analyses of concept drift and finally an in-depth misclassification and feature analysis. We conclude that caution is warranted when training ML models for malware analysis as concept drift and clear performance drops were observed even for models trained on larger datasets.

CCS Concepts: • **Security and privacy → Intrusion/anomaly detection and malware mitigation**; • **Computing methodologies → Artificial intelligence**.

Keywords: Malware classification, Control Flow Graphs, Concept drift, Temporal dissection, Threat analysis

1 Introduction

Today, malicious software is more disrupting than ever with more than 450,000 new malicious programs (malware) and potentially unwanted applications (PUA) being registered every day [1]. These malwares are frequently used for cyberattacks aimed at stealing information, deploying ransomware, or carrying out other illegal operations or cybercrimes. Furthermore, their versatility allows one to adapt and use them in a wide range of devices such as Internet of Things (IoT) and mobile domains [36, 46].

Modernization of security software and detection techniques have led hackers and malicious operators to think of new methods for bypassing firewalls, spreading malware through a system, and injecting backdoors [45]. This trend of building new malware variants based on techniques that are able to mislead “traditional" detection systems such as anti-virus scanners [11], has generated an increasing amount and diversity of malware applications. For this reason, it is not only important to detect malware but also to distinguish the (growing number of) different types of malware *families* in order to gain a better understanding of current and future malware capabilities and their possible impacts [17]. In this scenario, malware detection and classification has become a key objective within the field of information security [31].

In recent years, multiple machine learning (ML) algorithms have been proposed to detect malware elements. These models are usually trained to extract, analyze, and classify source programs as benign or malware [34]. Yet, the majority of studies in recent literature [15, 31] are purely focused on the classification task - typically racing towards the best classification performance while neglecting the interpretation of the results. More and more complex ML solutions are implemented to squeeze and alter the initially given information while common pitfalls are overlooked so that often over-optimistic conclusions are drawn that do not reflect real-world cybersecurity scenarios [5]. Furthermore, learning-based systems are typically evaluated in a “static" environment which does not represent the real world of constantly evolving malware [5]. In fact, program behaviour and malware families can change rapidly, leading trained ML models to become outdated after only a few months without being noticed [24]. Moreover, these learning models may also be subject to cyber-attacks aimed at deliberately “confusing" the model and altering prediction results, which in turn may cause financial loss, infrastructure damages, etc. [4]. This situation results in a loss of control over the original data, especially knowing that these models are implemented by ML experts that are neither the actual cybersecurity operators nor malware analysts [26].

Prior studies have tried to address these concerns [24, 44], for example proposing an approach to evaluate temporal decay on Android malware. However, the authors addressed

*F. Zola is also with the Public University of Navarre.
†This work was carried out while F. Zola was a visiting PhD student at University College London, Department of Computer Science.

the problem as a binary classification but they did not analyze the causes that generate the drift. Concept drift is also analyzed in [21] and [29]. In both cases, the authors only applied methods to detect concept drift, without analyzing the causes behind this performance decay and its relation with the malware behaviour.

For these reasons, we propose here a novel Cyber Threat Intelligence (CTI) approach for a portable executable (PE) multi-class task to shed light on the underlying issues with ML-based malware classification, investigate potential causes of model failure, and thus ultimately extract insightful information relevant to cybersecurity practitioners and analysts. CTI is a crucial step for gaining more knowledge about the input data evolution and the classifier behaviour [6], without limiting the conclusions to model performance. In this work, we perform a systematic study to analyze the given malware implementation (also called *behaviour*) and its evolution over time in order to detect potential threats and anomalies in the model behaviour.

Our methodology is based on three main steps. In Step 1, inspired by previous approaches in which embeddings were computed from malware Control Flow Graphs (CFGs) [40, 42], we propose to directly use graph structural properties of CFGs as embeddings for implementing a ML model for malware multi-class classification. The creation of a malware classifier is relevant for establishing common guidelines for the entire CTI process. Then, in Step 2, we analyze the temporal consistency and reliability of the implemented models to unveil concept drift and points of model failure. This step first applies a *temporal dissection* approach dividing the given data into short temporal chunks and using a rolling window to train multiple models with the rolling-window data in order to evaluate how the quality of the information affects model performance over time. Then, we also apply an additional *temporal aggregation* operation to take into account the amount of information, i.e., the number of temporal chunks used for training the models as well. Finally, in Step 3, we carry out an in-depth misclassification and feature analysis combining feature importance score, prediction results, and feature trend analysis to understand trends and patterns that may have generated the concept drift and model failure.

By applying CTI concepts and recently published guidelines [5], it is possible to extract new domain knowledge that could be used to enhance the analyst's expertise and to improve the detection of potential future attacks [10]. We believe that the introduced methodology can be applied in the training stage of a model to highlight behavioural trends of both models and input data. This information can help analysts create a more valid and transparent machine learning model. Furthermore, users and practitioners can acquire additional knowledge about the temporal consistency of their data and can use this information to determine when to update their model in order to prevent substantial performance decays. Finally, they can relate concept drift to specific input information, which may help devise precautionary actions to avoid model failure.

To the best of our knowledge, this is the first work that is not limited to merely analyzing PE malware multi-class classification performance, but proposes to use a detailed temporal analysis to detect and understand concept drift and points of failure in the context of CTI.

The main contributions of this work can be summarized as follows:

- We train a ML model able to classify 6 malware families (multi-class problem) using structural graph-properties obtained from malware CFGs;
- A *temporal dissection* operation is introduced to evaluate the quality of the training data and how it affects model performance over time (performance trend);
- We propose *temporal aggregation* to detect significant concept drift points, even when a larger dataset is used for training the model;
- An extensive misclassification and feature analysis is performed to extract insightful information regarding the causes that may have generated the found concept drifts and model failures.

The rest of the paper is organized as follows. In Section 2, concepts regarding malware classification and concept drift are introduced. In Section 3, we present the proposed methodology, whereas in Section 4, the dataset and the evaluation metrics considered, as well as the experiments configurations are introduced. Then, in Section 5, results are presented and discussions are reported in Section 6. Finally, Section 7 provides conclusions and guidelines for future work.

2 Background

In this Section, concepts about malware classification and concept drift are introduced. More specifically, Section 2.1 describes the malware classification task and its related works, whereas in Section 2.2 the concept drift problem is presented.

2.1 Malware Classification

Malware classification tasks are usually based on *static*, *dynamic*, or *hybrid* analyses. In the static case, malware binaries are analyzed without actually running the code. The code is used to extract information such as signatures, hashes, or for creating graph structures on top of which the classification is performed [39]. In dynamic analyses, the malware code is executed in an isolated environment and its behaviour is studied, as shown in [12]. However, this approach can be more complex and time-consuming due to the required malware execution in a secure environment. Finally, in the hybrid analysis, both static and dynamic information are combined for the final prediction [3]. One of the most promising approaches is based on static analysis, extracting CFGs in order to represent the flow of control between the basic blocks in

the program, i.e., a maximal-length sequence of a branch-free code [7]. In these CFGs, vertices represent sequential code without branches or jump targets, whereas edges represent the jumps in the control flow of the program.

Several works have made use of CFGs. In particular, Xu et al. [41] used CFGs and data flow graphs extracted from Android applications, whereas Xia et al. [39] proposed to combine CFGs with Term Frequency-Inverse Document Frequency for a malware family classification. A similar approach was explored in [38], where both semantic and structural features of CFGs were analyzed for the final classification. Yan et al. [42] proposed to use a deep graph convolutional neural network (DGCN) for exploiting structural embedding and basic block attributes of CFGs. They directly left the DGCN to learn how to transform graph information into embedding and used them for the final classification. A similar approach was introduced in Xu et al. [40], where embeddings were directly extracted from the CFGs using deep learning models and finally used for improving the malware similarity classification. However, since in both cases embeddings were computed directly by the learning models, they worked like "black boxes", which do not allow linking classification performance to the embedding values. In this sense, it was not possible to extract additional insightful information for improving CTI.

Inspired by the approaches presented in [40] and [42], in this work, we propose to directly use the structural properties of the CFGs as *graph embeddings* and use them for the final multi-class classification. In this way, our hypothesis is that we can directly relate classification performance to the structural graph embeddings and use the analysis of input trends to improve malware CTI. Furthermore, while [40] focused on firmware images or vulnerable functions for binary classification, we extract CFGs directly from a large PE malware dataset to perform a multi-class classification.

2.2 Concept Drift

In machine learning problems, results strongly depend on the data used during the training process. Hence, uncontrolled changes in the input features can generate inconsistent and misleading results. This problem becomes even more relevant when temporal data are analyzed and novel information is used as input for models trained on old data [33]. This problem is usually known as *concept drift* [37] or *dataset shift* [25]. An effective learner (*adaptive learner*) should be able to react to this problem, by detecting such changes and re-adjusting its predictions.

Concept drifts can appear following different temporal patterns. Usually, one can distinguish four categories [22]: *abrupt, incremental, gradual* or *reoccuring* drift. Concept drift is defined as abrupt when the change occurs instantaneously, for example, a sensor that breaks down. If performance changes are slowed in time and in values, they are considered incremental drifts. This is the case, for example, for a worn-out sensor. On the other hand, if the changes are only slowed in time but not in value, they are considered as gradual drift. Finally, if performance changes are repeated over a certain interval (e.g. seasonal), there may be reoccurring drifts, as in an external temperature sensor. It is to be noted that although outliers and noise can be seen as points that instantaneously change the data distribution, formally they are not considered as concept drift [22].

Concept drift is a highly relevant problem in cybersecurity applications, especially when machine learning models are implemented. It has been predominantly studied in applications such as anomaly detection [16], fraud [8] and spam [30] detection. However, in recent years, concept drift has been recognized as a key issue for malware detection as malware developers constantly try to create new variants to avoid detection, leading to continuously evolving malware behaviour that may render certain machine learning models outdated after a few months only. More specifically, in [20] and [9], authors try to improve classifier performance by detecting concept drift and creating adaptive models, i.e., systems able to detect drift while being in operation ("on-running") and address it by retraining themselves with more recent data. Other studies, such as [21] and [29], propose novel methods for detecting concept drift, however, they do not deeply analyze the relationship between these performance decays and the actual input of the malware classifier.

For these reasons, in this work, we propose to analyze concept drift in depth to improve our CTI analysis. Our objective is not to directly mitigate the concept drift, but to propose a method for studying it and for a better understanding of how it affects model decisions models over time.

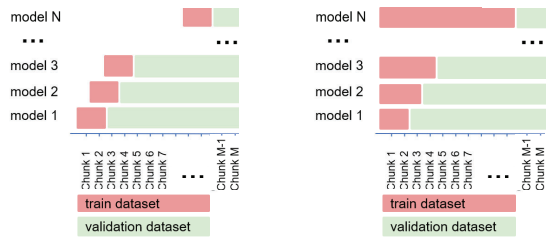
3 Towards practical malware threat analysis

In this work, a methodology for enhancing CTI related to PE malware is introduced and validated with a large binaries dataset. The idea of this approach is not only to propose a classification method for PE malware based on CFGs, but also to extract insightful information about the limitations of both the model and data. This methodology is based on three main steps: *Malware classification, Temporal analysis* and *Misclassification and feature analysis*. In the first step, malware binaries are converted into CFGs used for training ML models, whereas in second step a temporal analysis for unveiling concept drifts is performed. Finally, in the third step, misclassifications and feature trends are analyzed to discover possible causes of performance decays. The methodology steps are further detailed in the following sections.

Step 1 - Malware classification with CFG features. We first perform a *classification* task by extracting CFGs from malware binaries samples belonging to several malware families. Usually, malware binaries are shared "disarmed", i.e., they have several flags of their binary set to zero [18].

However, in order to analyze them properly, they need to be "rearmed". After that, we extract their CFGs by using the Angr tool[1]. Angr is a multi-architecture binary analysis toolkit, with the ability to perform both static and dynamic symbolic execution on binaries [28, 32]. Once the CFGs are extracted from each malware sample, we use several common graph properties to describe them. More specifically, the following 10 graph structural properties are extracted from each CFG: *number of nodes, number of edges, the graph is strongly connected (boolean) [23], number of strongly connected components [23], number of weakly connected components [35], number of isolated nodes, transitivity [19], maximum node degree, minimum node degree*, and *node degree on average*. These CFG properties allow us to transform the graph information into feature vectors (embedding) that are finally used for training ML classifiers.

Step 2 - Temporal Analysis. In this step, we propose to apply a *temporal analysis* based on two main concepts: *temporal dissection* and *temporal aggregation*. For temporal dissection, our idea is to evaluate how the *quality* of the data changes over time and how it affects the classification performance. In this sense, we propose to split the dataset into M temporal chunks of fixed size, and then train several ML models using a n-chunks rolling window data and finally evaluate them with data from the following chunks, respectively (Figure 1a). On the other hand, temporal aggregation operation changes the size of the training dataset to incorporate more chunks, as shown in Figure 1b. This strategy helps us to detect more consistent drifts points in the model performance, i.e., performance drops found when larger datasets are used during the model training. These drifts points are further analyzed in Step 3.



(a) Temporal dissection (two chunks rolling window data). **(b)** Temporal aggregation (training with different data amount).

Figure 1. Temporal analysis

Step 3 - Misclassification and feature analysis. In this step, we propose an in-depth analysis to understand possible causes for classification performance drops. In this sense, we firstly analyze confusion matrices to highlight which malware families are misclassified. Then, a feature importance analysis is carried out. This operation can be performed using different approaches, such as the mean decrease impurity (MDI) [27], permutation method [2], SHapley Additive exPlanation [14]. This approach helps us in ranking the features and focusing the analysis just on the most relevant CFG-related features used by the classifier. Finally, the temporal trends of these features, i.e., the feature values per class in all chunks, can be analyzed. In this way, it is possible to detect rare behaviours in the data that may help to explain what the model learned and why it misclassified certain classes.

[1]https://docs.angr.io/built-in-analyses/cfg

4 Experimental Framework

In this Section, we describe the dataset, the used metrics and the experiment configurations used in this work. More specifically, in Section 4.1 an overview about the used dataset is reported, whereas in Section 4.2 the metrics are described. Finally, in Section 4.3, we present the classifier and the parameters used during the experiments.

4.1 Dataset

In this study, a PE Malware dataset called SOREL-20M [18] is used. This dataset contains information about 20 million binaries, of which 10 million are benign and 10 million malicious, collected from January 1, 2017 to April 10, 2019 (28 months). The dataset publishes the executable files of the malware only, which are opportunely disarmed. For each of these malwares, a first-seen timestamp is provided as well as one or more labels related to its malware family (multi-label dataset). The dataset contains 11 malware families: *Adware, Dropper, Spyware, File Infector, Worm, Downloader, Flooder, Ransomware, Packed, Cryptominer* and *Installer*.

For the aim of this work, we focus only on the most represented classes which are Adware, Dropper, Spyware, Packed, File Infector, Worm, and Downloader. Furthermore, from this list, we exclude the Packed family as well, since the compression of the malicious code works as an obfuscation technique that obstacles their detection [43]. Hence, their identification is out of the scope of this work. This leaves us with 6 families to be distinguished.

As mentioned, malware samples in the SOREL-20M dataset can have more than one tag (multi-label problem), however, for the sake of simplicity, we focus on the ones that are characterized by one tag only, addressing the problem as a multi-class task. In order to decrease the computational cost mainly related to CFG extraction, 10,000 samples are randomly chosen among the obtained multi-class data to create the dataset for the experiments.

Dataset summary. The final dataset is composed of 60,000 single labelled malware samples (10,000 per class). All these samples are selected following the temporal distribution of the original SOREL-20M data (from January 2017 until April 2019, i.e., 28 months). Furthermore, one-month data will be used as a chunk for the temporal analysis, although

the chunk size may be variable across applications and the availability of past data.

4.2 Metrics.

F1-score, *Area Under the Receiver Operating Characteristic Curve (AUC-ROC or AUC)* and *Area Under Time (AUT)* are used to compare the performance of the different models. Hereby, *F1-score* represents the relation between actual positive labels and those given by the classifier; *AUC* represents a classifier's ability to distinguish between classes, and *AUT* indicates the area under the performance curve over time [24]. In our experiments, the AUT is computed using the F1-score monthly curve. Furthermore, two AUT variants are considered: AUT_{next6} and AUT_{last10}. The first one is computed considering only the 6 months following the ones used for training the model, whereas the latter is computed using only the last 10 months of the dataset. This is because the last 10 months of the dataset are the only months shared in all the validation datasets. For this reason, AUT_{last10} can be used for evaluating the performance of the models over a common dataset. Finally, AUC at 10% is also computed for comparing the models' abilities. This value represents the area under the ROC curve when the value of the false-positive rate (FPR) was fixed to 0.10. A good model should generate high values of the true positive rate for small values of FPR (high values of AUC at 10%).

4.3 Experiment Configurations

In all the experiments, a Random Forest (RF) classifier is used to learn and classify malware behaviours extracted from CFGs, since it has shown to perform well in similar tasks [13, 42]. In particular, in *Step 1 - Malware classification with CFG features*, a grid-search is performed in order to detect the best hyper-parameters for the RF model. More specifically, four models are implemented testing two values for the number of trees (100 and 1,000) and two for tree-depth (10 and 1,000). To do so, the initial dataset, composed of the graph properties (embeddings) extracted from each CFG, is divided into train, validation, and test datasets. This split is performed considering the temporal aspect, i.e., knowing that the whole dataset contains information of 28 months, the first 18 are used for creating the training dataset, the following 5 for composing the validation dataset and the last 5 months for creating the test dataset. These datasets are used for training, validating, and testing RF models.

For the *Step 2 - Temporal analysis*, the initial dataset is split into chunks of 1 month, generating 28 chunks (M). In particular, during the temporal dissection, a rolling window is applied to keep selecting 2 consecutive months (n) to be used for training a classifier, which is then evaluated with data from the following months - while moving forward in time, as described in Section 3. The process is repeated by moving the rolling window until reaching month 18, hence, generating 17 different models (N). These models followed the configuration of the ones that showed the best validation results in the previous experiment. On the other hand, during the temporal aggregation, 5 temporal sizes are used, i.e., 5 models are trained (N). In particular, train datasets from the first month until months 2, 4, 6, 12, and 18 are used. This approach simulates training with as much data as one has at hand in an equivalent real-world scenario, and it helps to detect consistent performance drifts.

Finally, in *Step 3 - Misclassification and feature analysis*, the MDI strategy is used for computing the feature importance, since it is widely used as feature importance in RF models [27]. The MDI value represents the sum of the gain associated with all the splits across all trees where the corresponding feature is used. Moreover, we focus our temporal analysis of the monthly feature trends to detect rare behaviours on the top 4 features.

5 Experimental Results

In this section, the results obtained by applying our methodology to the introduced malware dataset, are reported. In particular, in Section 5.1 the best RF configuration is identified (Step 1), whereas in Section 5.2, the temporal analyses are reported (Step 2). Finally, in Section 5.3, CTI results in the drop points are presented (Step 3).

5.1 Step 1 - Malware classification with CFG features.

As shown in Table 1, splitting the initial dataset following the temporal aspect generates train, validation and test datasets with 26,588, 18,152 and 15,260 samples, respectively. Table 1 shows that several malware families are used differently over time. Although the full dataset is balanced, several malware families like Dropper and Spyware have more samples in the validation dataset than in the training dataset, while the Worm family has more samples in the test dataset than in the train and validation sets.

	Short	Train	Validation	Test
Months from		01/2017	07/2018	12/2018
Months to		06/2018	11/2018	04/2019
Adware	Adw	6,062 (23%)	1,977 (11%)	1,961 (12%)
Dropper	Dro	3,645 (14%)	4,250 (23%)	2,105 (14%)
Spyware	Spy	3,515 (13%)	4,406 (24%)	2,079 (14%)
File Infector	Fil	6,693 (25%)	1,737 (10%)	1,570 (10%)
Worm	Wor	2,228 (8%)	2,952 (16%)	4,820 (32%)
Dowloader	Dow	4,445 (17%)	2,830 (16%)	2,725 (18%)
Total		26,588 (100%)	18,152 (100%)	15,260 (100%)

Table 1. Malware families distribution in the split dataset.

In Table 2, the results obtained by the RF models trained with different hyper-parameters in the validation dataset are reported. All models reach values of F1-score and AUC greater than the 82% and 96%, respectively. The results highlight that the depth of the tree is the parameter that allows for improving the performance of the classifier, whereas the

number of trees predominantly has an impact on the training time. The classifier trained with 1,000 trees (each one with a depth of 1,000), i.e., *RF4*, achieves the best results in the validation.

Models	# trees	depth tree	F1-score	AUC	AUC 10%	training time (s)
RF1	100	10	0.8291	0.9639	0.0826	**2.17**
RF2	1000	10	0.8225	0.9636	0.0824	21.65
RF3	100	1000	0.8537	0.9773	0.0871	2.91
RF4	1000	1000	**0.8652**	**0.9791**	**0.0875**	30.03

Table 2. Validation results for each trained model.

In Table 3, the results obtained by the RF4 on the validation and test dataset are compared. One can observe a decrease in model performance when the test dataset is used, especially in terms of F1-score (−0.1769). Therefore, this indicates that there is a strong decline when trying to generalize to new PEs that are far away in time. To better understand this behaviour, we will investigate model performance taking into account the temporal evolution of the data in the next experiment

5.2 Step 2 - Temporal analysis

Temporal dissection. In Figure 2, the monthly F1-scores obtained by the RF models trained with 2-months rolling window data are reported. Training the models with the early months in the dataset (from months 1 to 5) generates very poor results in terms of F1-score in almost every test month (Figure 2A). However, when months after 8 are used, the models show immediate high values in terms of F1-sore for the next 5 months until again breaking down on 18 (Figure 2B). Finally, training the models with data from 13 to 16 generate low performance scores (Figure 2C), while after these dates the models reach their best performances with F1-scores in a range of 0.60-0.90. Clear decay points, in which model performance broke down substantially, are observed for the months 12 and 19. These trends are confirmed by analyzing the overall AUT, the AUT_{next6}, and the AUT_{last10}, as shown in Table 4. Models trained on data from the early months (until months 14) show very low values in terms of overall AUT ($\leq$0.55) and AUT_{last10} ($\leq$0.52). On the other hand, several of them show to be promising in the 6 months immediately after those used in the training process, such as the model 10,11 ($AUT_{next6} \geq 0.77$). Results in Table 4 also highlight that something happened in months 6-7, as well as between months 12 to 14, since models trained with these data are not able to generalize the future, showing low values even in closest months ($AUT_{next6} \leq 0.56$). Yet, the best values are reached with the final months of data (training from month 16), with all the metrics $\geq$0.70.

Metrics	Validation	Test	GAP (Test-Validation)
F1-score	0.8652	0.6883	−0.1769
AUC	0.9791	0.8797	−0.0994
AUC 10%	0.0875	0.0555	−0.0320

Table 3. Validation and test results for the RF4 model.

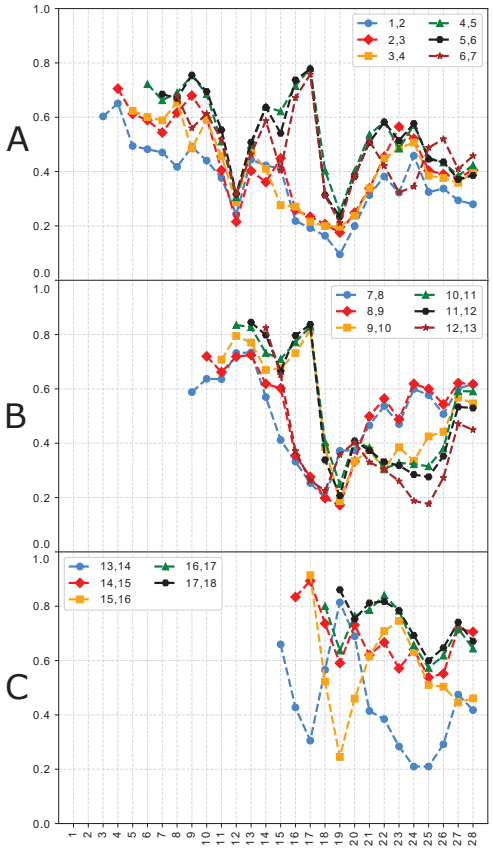


Figure 2. Monthly F1-score values for each model trained with 2-months rolling window data (temporal dissection).

Temporal aggregation. As shown in Table 5, the 2-months model achieves very low values in all the metrics ($\leq$0.53), whereas the 18-months model shows values $\geq$ 0.77. However, although the 12-months model is trained with a larger amount of data, when compared with models trained with only 2-months data (dissection models), it shows a lower performance with respect to several models. Nevertheless, testing months vary in this case. For this reason, Table 6 compares the difference in performance between the dissection models, i.e., models trained for only 2 months, and their respective models trained with all the data available until those months (aggregation models). Results show that models trained with 2-months data can reach competitive performance with respect to the models trained with larger datasets, especially in predicting the closest months. This is the case of 6-months and 12-months models that show

Model name	Training months	AUT	AUT_{next6}	AUT_{last10}
1, 2	1, 2 (2017)	0.3633	0.5215	0.3132
2, 3	2, 3 (2017)	0.4250	0.6110	0.3987
3, 4	3, 4 (2017)	0.4066	0.5871	0.3814
4, 5	4, 5 (2017)	0.5424	0.6815	0.4646
5, 6	5, 6 (2017)	0.5280	0.6348	0.4583
6, 7	6, 7 (2017)	0.4695	0.5163	0.4154
7, 8	7, 8 (2017)	0.5060	0.6636	0.5142
8, 9	8, 9 (2017)	0.5147	0.6770	0.5181
9, 10	9, 10 (2017)	0.5190	0.7262	0.3925
10, 11	10, 11 (2017)	0.5171	0.7747	0.3839
11, 12	11, 12 (2017)	0.4806	0.7379	0.3607
12, 13	12 (2017), 1 (2018)	0.3508	0.4204	0.3121
13, 14	1, 2 (2018)	0.4315	0.5575	0.3971
14, 15	2, 3 (2018)	0.6689	0.7356	0.6316
15, 16	3, 4 (2018)	0.5525	0.5309	0.5528
16, 17	4, 5 (2018)	0.7100	0.7640	0.7090
17, 18	5, 6 (2018)	0.7347	0.7889	0.7347

Table 4. Overall AUT, AUT_{next6} and AUT_{last10} for models trained with 2-months rolling window data (temporal dissection).

Model name	Training months	AUT	AUT_{next6}	AUT_{last10}
2-months	1 - 2	0.3679	0.5239	0.3166
4-months	1 - 4	0.4498	0.6529	0.4086
6-months	1 - 6	0.5547	0.6502	0.4960
12-months	1 - 12	0.6168	0.7489	0.5758
18-months	1 - 18	0.7701	0.8258	0.7701

Table 5. Overall AUT, AUT_{next6} and AUT_{last10} for models trained with different amount of data (temporal aggregation).

Aggregation models (A)	Dissection models (D)	AUT (A-D)	AUT_{next6} (A-D)	AUT_{last10} (A-D)
2-months	1,2	0.0046	0.0024	0.0046
4-months	3,4	0.0432	0.0658	0.0215
6-months	5,6	0.0267	0.0154	0.0377
12-months	11,12	0.1362	0.0110	0.2151
18-months	17,18	0.0354	0.0369	0.0354

Table 6. Difference between models trained with all the available data until a specific month (temporal aggregation - A) and models trained with only the two previous months (temporal dissection - D).

only an increase ≤ 0.016 in terms of AUT_{next6} with respect to models trained with just the last two months.

Figure 3 shows that each time new months are added to the training dataset, model performance increased. In fact, for each month, the F1-score of a model is always above the F1-score of the model trained with fewer data. Although the 18-months model generates promising values in terms of monthly F1-score, it is to be noted that the other models show highly varying trends depending on the considered

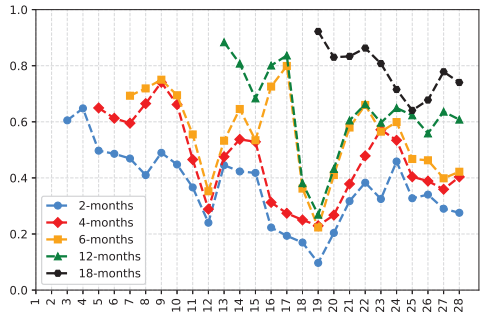


Figure 3. Monthly F1-score computed for models trained on differently sized temporal datasets (Step 2 - temporal aggregation)

date (Figure 3). In particular, 2- and 4-months models are affected by a *gradual* concept drift, since they show the first strong decline on month 12 and a consequent loss of performance (similar values) from months 14 to 21. On the other hand, 6- and 12-months models do not show a clear concept drift pattern even if, on specific dates, they show performance drops that in some cases can be considered both an *abrupt* concept drift or an *outlier*. Yet, it is interesting that even 12-months broke down in the months 18 and 19, which are determined as problematic months in the temporal dissection step as well.

For this reason, we propose to investigate these lower points (highest declines) in-depth, which are common to all models even when larger datasets are used for training them. In particular, the months 12, 18, and 19 in Figure 3 are further investigated, since in all three cases, models show an F1-score ≤ 0.40.

5.3 Misclassification and feature analysis

We start analyzing the misclassifications of the 2-, 4-, and 6-months models on month 12 (the complete confusion matrices can be found in the appendix, Figure 5). Although Adware, File Infector, and Downloader are detected pretty well, the main problem is generated by the Dropper samples which are classified as Spyware for the 2-months model (764 samples out of 841, approx. $\sim$ 91%), and as File Infector for 4- and 6-months models, confusing approximately the 94% and 92% of the samples, respectively. The same analysis is carried out for months 18 and 19 (the complete confusion matrices can be found in the appendix, Figure 6 and Figure 7). On both dates, 2-, 4-, and 6-months models tended to classify them as Adware with a rate between 92% and 97%, whereas 12-months model tended to label 72% and 93% of the samples as Downloader in months 18 and 19, respectively. Furthermore, all models show problems in detecting Spyware samples as well on both dates.

As introduced in Section 4.3, the MDI for each feature in all the trained models is computed (Table 7). Results show that

Mean Decrease Impurity (MDI)					
Features	**2-months**	**4-months**	**6-months**	**12-months**	**avg.**
# nodes	0.1448	0.1135	0.1065	0.1254	**0.1226**
# edges	0.1258	0.1086	0.1147	0.1242	0.1183
strongly conn.	0.0000	0.0000	0.0001	0.0001	0.0001
# weakly conn.	0.0819	0.1109	0.1162	0.0887	0.0994
# strongly conn.	**0.1599**	0.1377	0.1256	0.1295	**0.1382**
# isolated nodes	0.0552	0.0612	0.0696	0.0525	0.0596
transitivity	0.1095	**0.1576**	**0.1576**	**0.1477**	**0.1431**
# max. node degree	0.1217	0.1211	0.1239	0.1229	0.1224
# min. node degree	0.0709	0.0718	0.0721	0.0775	0.0731
# avg. node degree	0.1305	0.1177	0.1137	0.1315	**0.1234**

Table 7. Feature importance scores based on MDI analysis. The most important feature for each model are highlighted, as well as the top-4 on average.

4-, 6-, and 12-months models are highly aligned, producing very similar importance scores for each feature. In particular, for these models, the most important feature is *transitivity*, whereas for the 2-months model the most important feature is *number of strongly connected components*. Averaging the importance score generated by all the models for each feature, the 4 most important features are *transitivity, number of strongly connected components, average degree of nodes* and *number of nodes*.

The temporal monthly trends of these top 4 features are extracted and compared in order to detect rare behaviours in the data that may help to explain what the model has learnt and why it misclassifies certain classes. In particular, the 4 feature trends of Dropper, Spyware and File Infector samples are analyzed until month 12 (Figure 4a) as these classes are confused for this data. We observe that the transitivity in the Dropper samples abruptly changes its average value in months 11 and 12, reaching values never shown before those dates. Furthermore, it is possible to appreciate also a change in the trend of Dropper samples in terms of average node degree and number of nodes (Figure 4a). In fact, in both features, the Dropper elements have different behaviour in the early months with respect to the other families. However, in months 11 and 12 they change it to a similar behaviour usually shown by Spyware and File Infector families.

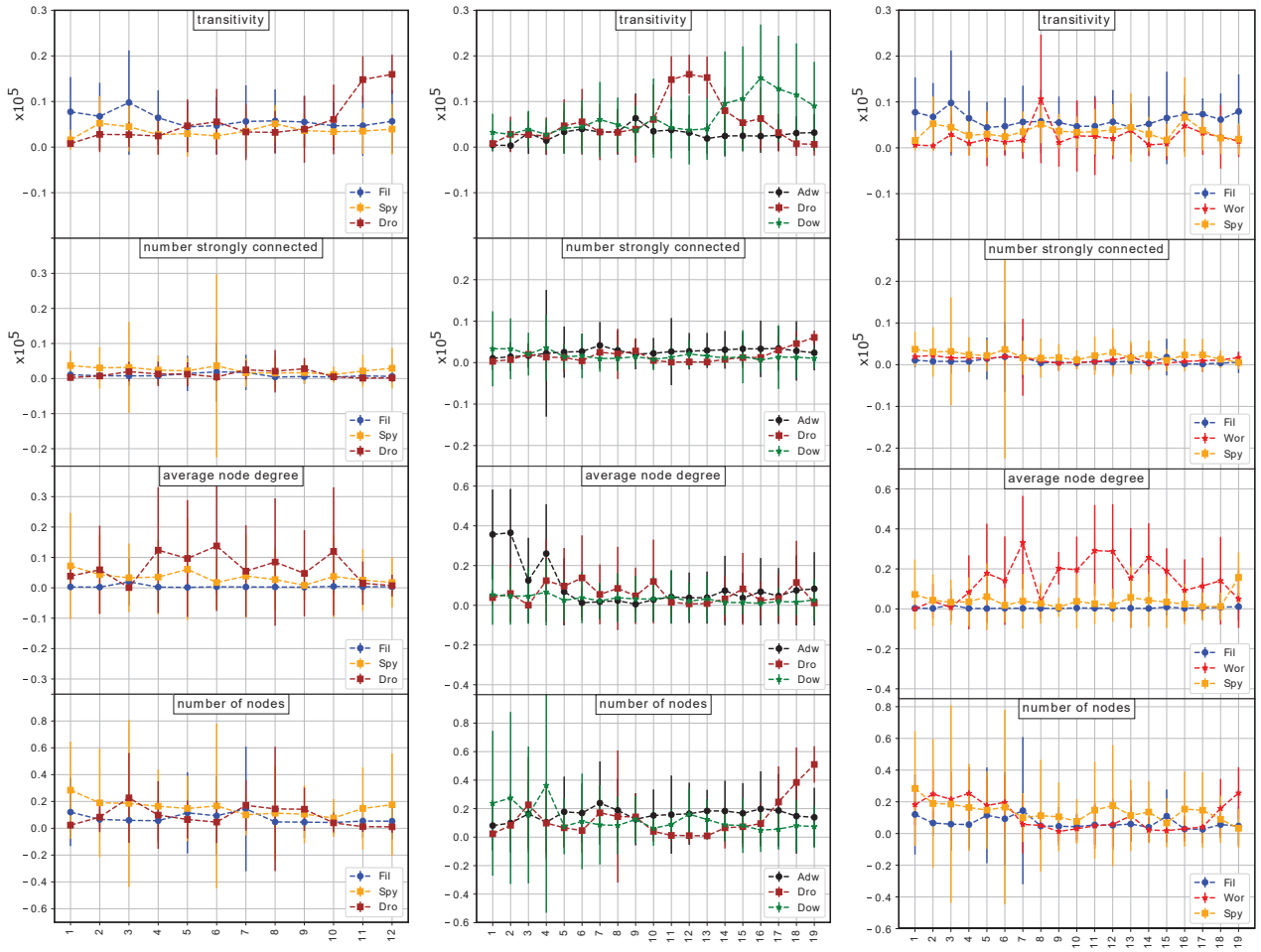
In Figure 4b and Figure 4c, the trends of the features are analyzed until months 19. In particular, from Figure 4b, it is possible to explain why models misclassify Dropper samples by analyzing the trends in the number of strongly connected components and the number of node features. In fact, in both cases, the trend of the Dropper samples follows low values in the early months, always superseded by the values of the other two classes, while, after month 17, the Dropper values surpass the other two classes. Furthermore, in terms of transitivity, the Dropper shows a highly varying trend, ending in months 18 and 19 at close-to-zero values.

Regarding Spyware misclassification, as shown in Figure 4c, the Spyware transitivity reaches the same values as the Worms family in months 17, 18, and 19. A similar behaviour occurs for the number of nodes feature, in months 18 and 19, in which the Spyware trend decreases reaching File Infector values. In terms of average node degree, the Spyware samples have always shown very low values, following the trend of the Worm family, however, on month 19 they change abruptly showing a peak.

6 Discussion

The temporality of the data and hence the lifetime of malware has strongly affected model performance. This is highlighted by a substantial decrease in the classification performance comparing results for the validation and the test datasets reported in Table 3. The same trend is confirmed by the *temporal dissection* analysis we have carried out (Figure 2). Analyzing the performance of models trained with the presented rolling window technique one can appreciate that all models work better (and surprisingly well) in months closer to the months used for training them, whereas their performance decays dramatically when evaluating with far away temporal data, i.e., malware developed about 5 months later. This is an important finding as it warrants caution when applying once-trained models in real-world applications: especially when dealing with malware, models do have a "lifetime" and should be re-trained after defined intervals. On the other hand, *temporal aggregation* strategy has shown that the more data are used in the training dataset, the better is the overall classifier performance (Table 5). However, comparing the results of these models in terms of AUT_{next6} with the ones obtained using temporal dissection models (Table 6), it is to be noted that in several cases, they can be competitive, i.e., including more data into the training process do not generate substantial improvements. For this reason, having a larger period for analysis could help understanding which is the right amount of data to be considered, since it may always not be beneficial to consider too old data. Furthermore, temporal aggregation analysis has shown that even using a larger dataset in the training process, all the considered models are affected by concept drift. In particular, 2- and 4-months models in Figure 3 show several clear points of decline after which their monthly performance tended to decrease. The other two models (6- and 12-months models in Figure 3) do not show a clear concept drift pattern but more occasional drops which can be considered both abrupt concept drift or outliers.

In the final misclassification and feature analysis step, we first analyze the confusion matrices extracted for the dates of critical model failure. A misclassification in two consecutive months, of the same malware families (Dropper and Spyware) may be a symptom of something changing in the malware implementation, which lead their CFG similar to other malware families. Table 7 shows that all structural graph properties extracted from the CFG are useful for the final classification, excluding the property of "graph is strongly connected", which can be removed in future applications to



(a) File Infector, Spyware and Dropper until month 12 **(b)** Adware, Downloader and Dropper until month 19 **(c)** File Infector, Worm and Spyware until month 19

Figure 4. Trends of the top 4 features for the confused classes until months 12 and 19.

improve generalization and computational efficiency. This analysis is important to understand which properties are the most relevant for the classification, and so to detect which ones represent the most prone to be used in an attack. In fact, a slight change in their values can easily generate misclassification. When analyzing the trends of the top-4 features (Figure 4), we observe that usually there is at least one feature that changes substantially in the months affected by the performance decays. At the same time, other features seem to line up with features from the classes that the initial class is confused with. Both phenomena can favour the misclassification.

Limitations. The three-step methodology proposed and developed in this work has shown promising results when applied to a relevant subset of the SOREL-20M dataset in the context of multi-class PE classification. However, there are some limitations to be considered. In this study, the malware analysis is performed using CFGs extracted using external tools (Angr). It should be considered that if this tool is no longer available, supported, or improved, classification results may change. A further drawback is that even though changes in feature trends are directly related to the graph topology and its structure, they cannot currently be used to identify a modification of the malware binary to highlight a specific routine.

Relation with common pitfalls. In order to present the quality of our methodology, we discuss here possible pitfalls that can generate misinterpretation of research results. These pitfalls are analyzed using the same notation as introduced in [5].

P1. Samples Bias and *P2. Label Inaccuracy* represent the hardest pitfalls to mitigate. In fact, although we have used a single dataset for our analysis (SOREL-20M), it is composed of information from multiple sources (P1), whereas, we could

not verify the correctness of all the labels (P2). For this reason, on the one hand, we accept the limitations of the dataset (P1), being aware of the possible bias that it can introduce. On the other hand, we have focused the analysis only on samples with a single label (P2).

P3. Data Snooping, *P4. Spurious Correlations* and *P5. Biased Parameter Selection* represent three pitfalls related to the system design. P3 and P5 have been mitigated by splitting the dataset into train, validation, and test sets for validating the performance. On the other hand, regarding P4, we have provided analysis about feature importance to highlight which features are more relevant for the classification, becoming more prone to be used in an attack.

P6. Inappropriate Baseline, *P7. Inappropriate Performance Measures* and *P8. Base Rate Fallacy* are common pitfalls related to the evaluation of the performance. In our approach, the models themselves evaluated over time have been used as the baseline models (P6). However, following suggestions given for P7, appropriate performance measures are used, such as AUC, AUT, and F1-score. Regarding P8, the overall dataset is balanced, since we have selected the same amount of samples for each class. However, classes are not balanced temporally due to their distribution in the original dataset.

Finally, the last two pitfalls - *P9. Lab-Only Evaluation* and *P10. Inappropriate Threat Model.* – are related to the deployment in real scenarios. In this work, we have dealt with these two pitfalls trying to replicate real-world conditions, such as considering data availability in time, the evolution of behaviours, the amount of training information (which can affect the model usability), etc. At the same time, we have tried to outline recommendations for practitioners and researchers which help to understand decisions predicted by ML models, and that may help to draw conclusions in real applications.

7 Conclusions and Future work

With this work, we aimed to highlight the problems that may arise when applying ML-based multi-class classification to real-world malware data and proposed a three-step approach to carry out an in-depth analysis for Cyber Threat Intelligence (CTI). First, we extracted Control Flow Graphs (CFGs) from a large multi-class malware dataset and trained a classifier in the traditional, static way. Then, we performed a detailed temporal analysis of concept drifts, in which models were trained a) using data from 2-months rolling windows (temporal dissection) and b) by gradually increasing the amount of data available for training (temporal aggregation). Finally, in the third and last step, performance breakdowns related to concept drift are further analyzed. The outcomes of this study can be summarized as follows:

- Malware families can be correctly classified using structural graph properties extracted from Control Flow Graphs (CFGs);
- Our *temporal dissection* approach highlighted that several models trained with only 2 months of recent data performed surprisingly well on immediately following data;
- Concept drift and several distinct points of model failure could be observed even using models trained on a relatively large amount of data;
- Analyzing trends of the most important features over time, in the lead-up to critical points of model failure it was usually due to one feature changing substantially and rather abruptly.

Our final conclusion is that malware features as captured by structural graph properties from CFGs are constantly changing and evolving over time leading to significant concept drifts and points of classifier failure (in our case in particular for Dropper and Spyware classes). Therefore, training a model and leaving it in operation without re-training presents a potentially serious security risk for practitioners and stakeholders. Following our results, it may be beneficial to train models on fewer but recent data, apply them for a few months only and then re-train using the newest data again (rolling-window approach) - even though further research is necessary to confirm this hypothesis and to better define re-train intervals.

In further future work, it may be interesting to focus the analysis on validating the results via graph machine learning techniques. Such algorithms are able to process graph information without extracting graph properties. Furthermore, all models implemented here showed concept drift and most showed distinct points of model failure. This hints that the problem of malware multi-class classification indeed is not resolved yet and that other strategies need to be explored. Finally, in terms of interpretation of the results, one could link the behaviours detected in the graph properties like transitivity, number of strongly connected components etc., with the actual malware binary (code), and thus directly link graph behaviour with the part of the code that has generated such change. Through this approach, cyber analysts could apply precautionary actions to promptly detect and mitigate concept drift, avoiding model failure.

Acknowledgments

This work has been partially supported by the Spanish Centre for the Development of Industrial Technology (CDTI) under the project ÉGIDA (EXP 00122721 / CER-20191012).

References

[1] 2022. AV-TEST. Malware Statistics and Trends Report by AV-TEST. Retrieved June 11, from. https://www.av-test.org/en/statistics/malware/

[2] André Altmann, Laura Toloşi, Oliver Sander, and Thomas Lengauer. 2010. Permutation importance: a corrected feature importance measure. *Bioinformatics* 26, 10 (2010), 1340–1347.

[3] Blake Anderson, Curtis Storlie, and Terran Lane. 2012. Improving malware classification: bridging the static/dynamic gap. In *Proceedings of the 5th ACM workshop on Security and artificial intelligence*. 3–14.

[4] Eirini Anthi, Lowri Williams, Matilda Rhode, Pete Burnap, and Adam Wedgbury. 2021. Adversarial attacks on machine learning cybersecurity defences in industrial control systems. *Journal of Information Security and Applications* 58 (2021), 102717.

[5] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2022. Dos and don'ts of machine learning in computer security. In *Proc. of the USENIX Security Symposium*.

[6] Mauro Conti, Tooska Dargahi, and Ali Dehghantanha. 2018. Cyber threat intelligence: challenges and opportunities. In *Cyber Threat Intelligence*. Springer, 1–6.

[7] Keith Cooper and Linda Torczon. 2011. *Engineering a compiler*. Elsevier.

[8] Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. 2015. Credit card fraud detection and concept-drift adaptation with delayed supervised information. In *2015 international joint conference on Neural networks (IJCNN)*. IEEE, 1–8.

[9] Abdulbasit A Darem, Fuad A Ghaleb, Asma A Al-Hashmi, Jemal H Abawajy, Sultan M Alanazi, and Afrah Y Al-Rezami. 2021. An Adaptive Behavioral-Based Incremental Batch Learning Malware Variants Detection Model Using Concept Drift Detection and Sequential Deep Learning. *IEEE Access* 9 (2021), 97180–97196.

[10] Ali Dehghantanha, Mauro Conti, Tooska Dargahi, et al. 2018. *Cyber threat intelligence*. Springer.

[11] Jérémy Donadio, Guillaume Guerard, and Soufian Ben Amor. 2021. Collection of the Main Anti-Virus Detection and Bypass Techniques. In *International Conference on Network and System Security*. Springer.

[12] Ying Fang, Bo Yu, Yong Tang, Liu Liu, Zexin Lu, Yi Wang, and Qiang Yang. 2017. A new malware classification approach based on malware dynamic analysis. In *Australasian Conference on Information Security and Privacy*. Springer, 173–189.

[13] Parvez Faruki, Vijay Laxmi, Manoj Singh Gaur, and P Vinod. 2012. Mining control flow graph as api call-grams to detect portable executable malware. In *Proceedings of the Fifth International Conference on Security of Information and Networks*. 130–137.

[14] Daniel Fryer, Inga Strümke, and Hien Nguyen. 2021. Shapley values for feature selection: The good, the bad, and the axioms. *IEEE Access* 9 (2021), 144352–144360.

[15] Ekta Gandotra, Divya Bansal, and Sanjeev Sofat. 2014. Malware analysis and classification: A survey. *Journal of Information Security* (2014).

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

[17] Steven Strandlund Hansen, Thor Mark Tampus Larsen, Matija Stevanovic, and Jens Myrup Pedersen. 2016. An approach for detection and family classification of malware based on behavioral analysis. In *2016 International conference on computing, networking and communications (ICNC)*. IEEE, 1–5.

[18] Richard Harang and Ethan M. Rudd. 2020. SOREL-20M: A Large Scale Benchmark Dataset for Malicious PE Detection. arXiv:2012.07634 [cs.CR]

[19] Paul W Holland and Samuel Leinhardt. 1971. Transitivity in structural models of small groups. *Comparative group studies* 2, 2 (1971), 107–124.

[20] Donghui Hu, Zhongjin Ma, Xiaotian Zhang, Peipei Li, Dengpan Ye, and Baohong Ling. 2017. The concept drift problem in Android malware detection and its solution. *Security and Communication Networks* (2017).

[21] Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting concept drift in malware classification models. In *26th USENIX Security Symposium (USENIX Security 17)*. 625–642.

[22] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. 2018. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering* 31, 12 (2018), 2346–2363.

[23] Esko Nuutila and Eljas Soisalon-Soininen. 1994. On finding the strongly connected components in a directed graph. *Information processing letters* 49, 1 (1994), 9–14.

[24] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 729–746.

[25] Joaquin Quiñonero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. 2008. *Dataset shift in machine learning*. Mit Press.

[26] Sherif Saad, William Briguglio, and Haytham Elmiligi. 2019. The curious case of machine learning in malware detection. *arXiv preprint arXiv:1905.07573* (2019).

[27] Erwan Scornet. 2020. Trees, forests, and impurity-based variable importance. *arXiv preprint arXiv:2001.04295* (2020).

[28] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Audrey Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. 2016. SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. (2016).

[29] Anshuman Singh, Andrew Walenstein, and Arun Lakhotia. 2012. Tracking concept drift in malware families. In *Proceedings of the 5th ACM workshop on Security and artificial intelligence*. 81–92.

[30] Ge Song, Yunming Ye, Haijun Zhang, Xiaofei Xu, Raymond YK Lau, and Feng Liu. 2016. Dynamic clustering forest: an ensemble framework to efficiently classify textual data stream with concept drift. *Information Sciences* 357 (2016), 125–143.

[31] Alireza Souri and Rahil Hosseini. 2018. A state-of-the-art survey of malware detection approaches using data mining techniques. *Human-centric Computing and Information Sciences* 8, 1 (2018), 1–22.

[32] Nick Stephens, John Grosen, Christopher Salls, Audrey Dutcher, Ruoyu Wang, Jacopo Corbetta, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. 2016. Driller: Augmenting Fuzzing Through Selective Symbolic Execution. (2016).

[33] Alexey Tsymbal. 2004. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin* 106, 2 (2004), 58.

[34] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. 2019. Survey of machine learning techniques for malware analysis. *Computers & Security* 81 (2019), 123–147.

[35] JJP Veerman and Ewan Kummel. 2019. Diffusion and consensus on weakly connected directed graphs. *Linear Algebra Appl.* 578 (2019).

[36] Aohui Wang, Ruigang Liang, Xiaokang Liu, Yingjun Zhang, Kai Chen, and Jin Li. 2017. An inside look at IoT malware. In *International Conference on Industrial IoT Technologies and Applications*. Springer.

[37] Gerhard Widmer and Miroslav Kubat. 1996. Learning in the presence of concept drift and hidden contexts. *Machine learning* 23, 1 (1996).

[38] Bolun Wu, Yuanhang Xu, and Futai Zou. 2021. Malware Classification by Learning Semantic and Structural Features of Control Flow Graphs. In *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 540–547.

[39] Rongze Xia and Baojiang Cui. 2021. Malware Classification Based on Graph Neural Network Using Control Flow Graph. In *International Conference on Broadband and Wireless Computing, Communication and Applications*. Springer, 129–138.

[40] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. 2017. Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 363–376.

[41] Zhiwu Xu, Kerong Ren, Shengchao Qin, and Florin Craciun. 2018. CDGDroid: Android malware detection based on deep learning using CFG and DFG. In *International Conference on Formal Engineering Methods*. Springer, 177–193.

[42] Jiaqi Yan, Guanhua Yan, and Dong Jin. 2019. Classifying malware represented as control flow graphs using deep graph convolutional neural network. In *2019 49th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 52–63.

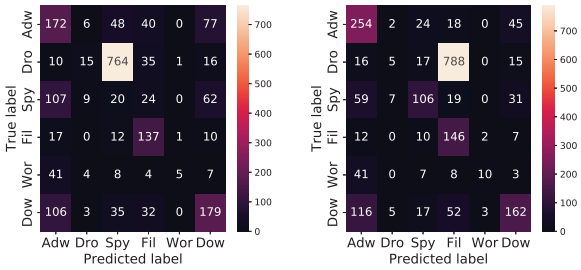
[43] Wei Yan, Zheng Zhang, and Nirwan Ansari. 2008. Revealing packed malware. *ieee seCurity & PrivaCy* 6, 5 (2008), 65–69.

[44] Gaofeng Zhang, Yu Li, Xudan Bao, Chinmay Chakarborty, Joel JPC Rodrigues, Liping Zheng, Xuyun Zhang, Lianyong Qi, and Mohammad R Khosravi. 2022. TSDroid: A Novel Android Malware Detection Framework Based on Temporal & Spatial Metrics in IoMT. *ACM Transactions on Sensor Networks (TOSN)* (2022).

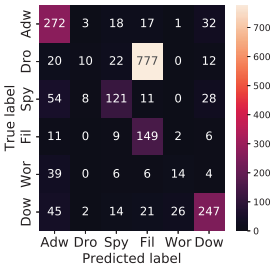
[45] Qinghua Zhang and Douglas S Reeves. 2007. Metaaware: Identifying metamorphic malware. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. IEEE, 411–420.

[46] Saman Zonouz, Julian Rrushi, and Stephen McLaughlin. 2014. Detecting industrial control malware using automated plc code analytics. *IEEE Security & Privacy* 12, 6 (2014), 40–47.

A Confusion matrices

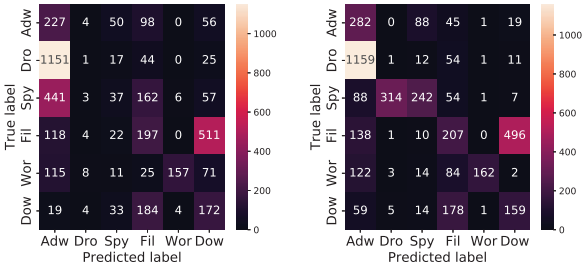


(a) 2-months model **(b)** 4-months model

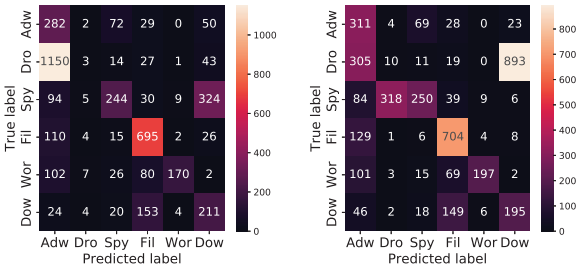


(c) 6-months model

Figure 5. Confusion matrices of the 2-, 4-, and 6-months models computed using test data on month 12.

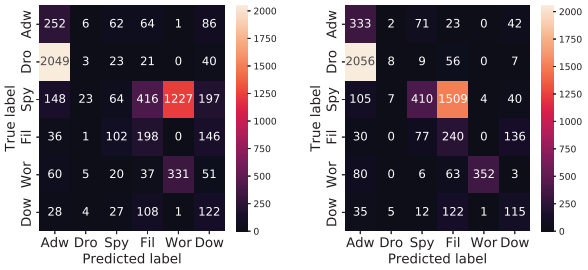


(a) 2-months model **(b)** 4-months model

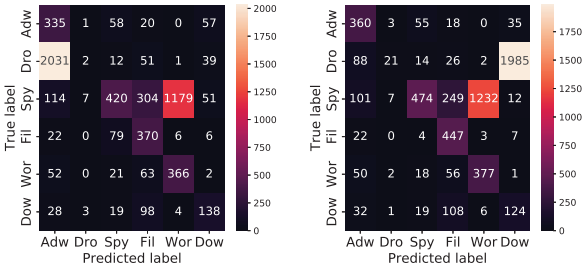


(c) 6-months model **(d)** 12-months model

Figure 6. Confusion matrices of the 2-, 4-, 6-, and 12-months models computed using test data on month 18.



(a) 2-months model **(b)** 4-months model



(c) 6-months model **(d)** 12-months model

Figure 7. Confusion matrices of the 2-, 4-, 6-, and 12-months models computed using test data on month 19.