TECHNISCHE
UNIVERSITÄT
WIEN
Vienna | Austria

# Machine Learning for Network Traffic Analysis

## Feature Spaces and Model Optimization

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

**Doktor der Technischen Wissenschaften**

by

**Dipl.-ing. Fares Meghdouri, Bsc.**
Registration Number 01617933

to the Faculty of Electrical Engineering and Information Technology

at the TU Wien

Advisor: Univ. Prof. Dipl.-Ing. Dr.-Ing. Tanja Zseby
Co-Advisor: Dr.techn. Felix Iglesias Vazquez, Msc.

The dissertation has been reviewed by:
A.o. Prof. Dr. Anna Sperotto
Prof. Dr.-Ing. Georg Carle

Vienna, 20th September, 2023

# Erklärung zur Verfassung der Arbeit

Dipl.-ing. Fares Meghdouri, Bsc.

Hiermit erkläre ich, dass die vorliegende Arbeit gemäßdem Code of Conduct — Regeln zur Sicherung guter wissenschaftlicher Praxis (in der aktuellen Fassung des jeweiligen Mitteilungsblattes der TU Wien), insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In— noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, 20. September 2023

# Acknowledgements

**ACK no.1** The Titan Xp GPU used for this research was donated by the NVIDIA Corporation.

**ACK no.2** Part of this work was supported by the project MALware cOmmunication in cRitical Infrastructures (MALORI), funded by the Austrian security research program KIRAS of the Federal Ministry for Agriculture, Regions and Tourism (BMLRT) under grant no. 873511.

*To you, Mother and Father.*

# Abstract

Machine Learning (ML) has revolutionized the field of network traffic analysis and anomaly detection, providing promising and efficient methods for predicting and defending against cyber threats. Traditional systems frequently rely on manual inspection and rule-based techniques, which have difficulties detecting zero-day attacks and coping with encrypted traffic, among other issues. In contrast, ML enables the creation of dynamic and adaptive systems that continuously adjust and learn in response to changing settings automatically. However, the application of ML in network traffic analysis is still far from expectations raised in the last decades and introduces new challenges, particularly in relation to network traffic data. Two key aspects include: (a) designing highly discriminative traffic representations in the presence of encrypted network traffic, and (b) improving the accuracy of algorithms while maintaining computational feasibility.

In this thesis, I address both of these issues. Specifically, I (a) propose new network traffic representations that can effectively handle traffic encryption and (b) enhance prevalent detection and classification architectures by incorporating novel concepts that improve training, performance, as well as time complexity. More specifically, in the first part of the thesis I present three new network traffic representations: the Multi-key feature vector, a cross-layer representation that allows significantly overcoming state of the art attack detection; I-Notice, a novel approach for handling unaggregated packet sequences; and FlowCount, a method for counting flows in IPsec tunnels. I also explore the effects of data scaling, as well as the importance of data transformation for training speed, model stability, and interpretability. In the second part of the thesis, I present three methods for improving ML techniques when applied to traffic classification and anomaly detection, namely: ODM, a sampling algorithm that provides coresets for faster and more accurate training; FlowGan, a framework for solving data imbalance and controlling network traffic generation; and EagerNet, a strategy for speeding up fully-connected neural network training while using fewer resources.

I conduct a thorough performance and explainability analysis whereupon results show that the proposed methods are highly effective in improving analysis accuracy and speed while overcoming limitations set by traffic encryption. Overall, the advances presented in this thesis cover critical aspects necessary to address the requirements of modern network traffic analysis.

# Contents

# Part I

# Machine Learning for Network Traffic Analysis

CHAPTER 1

# Introduction

## 1.1 Overview

In this chapter, I provide a brief overview on Network Traffic Analysis (NTA), anomaly detection and classification. I briefly discuss the fundamental concept underlying current methods and discuss the problem statement addressed in this thesis.

### 1.1.1 Network Traffic Classification and Anomaly Detection

Over the years, the evolution and expansion of the Internet has increased the need for surveillance of communication networks, particularly the traffic exchanged between various nodes around the globe [13, 31, 32]. Monitoring and analyzing network traffic are inevitable, whether for improving user experience, solving infrastructure problems, or detecting attacks and alerting parties in the event of abnormal behavior. Researchers began developing methods and systems to monitor communication networks for various purposes shortly after establishing functional networks [70]. This facilitated the advancement of infrastructure development and continues to assist current operators in the progression and protection of their infrastructures. This is where NTA comes into play. It is the art of investigating communication network data observed at a specific point in the network and for a specific period. Then, extracting useful information and using it for assessing performance, predicting communication phenomena, the reaction in case of failures, and improvement of infrastructures and user experience.

Network traffic classification is a sub-domain of NTA and refers to predicting the class of each network observation (typically a packet or a flow), where a class refers to a type, protocol, or service such as web browsing, file exchange, instant messaging, etc. The classification is used for several purposes, including automated filtering, Quality of Service (QoS), and other applications. For many years, classification was mostly accomplished through rule-based methods, e.g., by matching the protocol number, transport port numbers, or the payload (Deep Packet Inspection (DPI)) to a pre-stored set of signatures

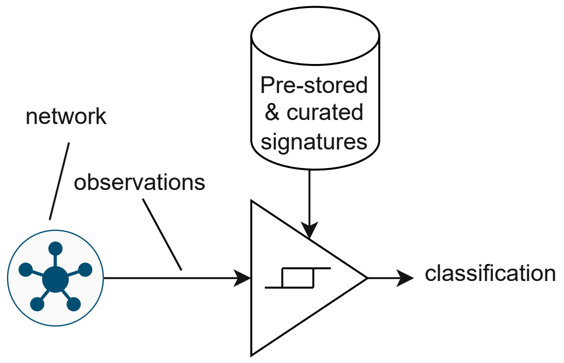


**Figure 1.1:** A basic signature-based network traffic classification scheme.

(e.g., "QoS Monitoring" from Cisco[1]). A signature-based traffic classification scheme is illustrated in Figure 1.1. First, a sample is collected, processed then compared to a set of carefully selected signatures; a class is afterward determined based on a similarity threshold (could be a total match or partial match). In most cases, manual input from domain experts is required to create the signatures in the first place. For example, a simple classification determines whether a packet is part of an Hypertext Transfer Protocol (HTTP) exchange. In this case, one could check if the packet's protocol is Transmission Control Protocol (TCP) and its destination or source port is 80. If so, an educated guess would be that this packet is genuinely a part of an HTTP exchange. However, with the advancement of network protocol design and the increase in traffic encryption and aggregation, conventional techniques (e.g., signature-based) have become less viable [56, 63], necessitating the use of more versatile and "intelligent" tools for information extraction. Such tools allow for easier differentiation between classes and achieve comparable, if not better, classification and detection performance than the previously discussed methods.

In the same context, network anomaly detection can be described as a sub-category of network traffic analysis and classification. It covers detecting any network activity that deviates from normal behavior, resulting in an anomaly. This definition in fact can be applied to various domains but also to ours [48]. It could be identified by detecting a change in behavior, such as an increase in the number of requests, matching, such as finding a known malware/virus signature, or simply observing an unauthorized application exchange, such as a Webserver exchanging Secure SHell (SSH) traffic. It is also worth noting that classification and anomaly detection can be performed concurrently, for example, classifying traffic into a given network application, such as HTTP, while also detecting anomalous behavior, e.g., brute-forcing a login page. Anomaly detection has been performed for many years using one of the methods above. However, the evolution of threats necessitates the development of sophisticated tools that establish an "understanding" of the attack's profile and the usage of statistical models capable of

[1]`cisco.com/c/en/us/support/ios-nx-os-software/ios-xe-16/series.html`

grasping complex network traffic patterns while performing faster than simple signature matching. ML (either supervised, unsupervised, or reinforcement learning) addresses such challenges by providing a variety of analysis methods.

### 1.1.2 Machine Learning for Network Traffic Analysis

ML is the task of using specific algorithms to extract “knowledge” from large sets of data and represent it as a mathematical model [179]. Arthur Samuel, an IBM computer scientist, introduced the term for the first time in 1959 [211]. There has been since a tremendous amount of work done in this field, and numerous applications have benefited from the utility of such powerful models to solve various problems, such as data mining, convex optimization and modeling [127]. The availability of large datasets and computing power in recent years has aided further in the widespread adoption of ML. This availability of large amounts of network stream data allowed ML to become a better choice for information extraction from network observations, thus automating a task that domain experts have manually performed. This does not imply that corrections, tuning, and manual inputs are not required anymore; instead, they are always utilized to tweak obtained models and make crucial decisions.

The integration of ML in existing solutions greatly aided traffic classification, anomaly detection, and many other subfields of NTA to simplify the task and achieve comparable performance to previous proposals [226, 142]. However, as discussed throughout this thesis, most existing approaches require adaptation for this integration to function, such as particular data preprocessing for a successful application of the learning model.

However, there is a cost to using ML. While algorithms can extract knowledge from data, it is sometimes unclear what extracted models have learned, which raises trust concerns, particularly in sensitive applications [188]. As a result, there is a tremendous amount of ongoing research in developing tools to explain the behavior of black-box models [262]. Among other domains, eXplainable Artificial Intelligence (XAI) is fundamental for NTA and must be considered carefully before developing any tool. Communication networks are essential to today’s infrastructure and its sustainability; hence, having arbitrary and unexplained alarms or actions can lead to undesirable consequences.

Further discussion on ML and how it relates to our field is presented in Chapter 2.

### 1.1.3 Feature Engineering for Network Traffic

An important step when exploiting ML is the approach for selecting and transforming collected data to reveal patterns and information in every instance better. This helps algorithms discriminate between classes and better grasp the raw data. This set of operations is typically referred to as **feature engineering**. In other words, feature engineering is the process of mapping the data from its original form into a mathematical space that allows for maximum discrimination between attributes. In NTA, we often create features by combining lower-level attributes (i.e., directly from observed traffic), which is beneficial for explainability analysis, as discussed in Part II.

In NTA, data is conventionally conceptualized as a series of packets transmitted across a network. At the most fundamental level, a packet consists of a sequence of bits, which can be clustered into meaningful chunks (fields). To derive a set of discriminant features, it is necessary to understand each packet segment and subsequently map them onto a suitable representation, such as numerical, categorical, or alternative formats. Once this transformation is accomplished, the information can be extracted, manipulated, or combined to generate a synopsis of network activities.

The efficacy of ML-based NTA is heavily reliant on feature engineering whereby a well-designed process results in a more effective representation of the data, which, in turn, facilitates data processing. Conversely, inadequate feature engineering may impede the success of even the most sophisticated algorithms [75]. To address this issue, I delve into feature engineering in more detail in Part II, and provide insights into how it can be enhanced.

## 1.2 Motivation

The primary motivation behind this thesis is to address the critical need for security to protect and defend against the increasing number of threats in modern networks. Moreover, while most research endeavors aim to answer the question of "How to solve problem X?" I focus on a different yet equally crucial question: "What are the limitations of solution Y, and how can they be surmounted to solve problem X better and faster?". An extensive literature review on the combination of ML and NTA revealed that many researchers employ similar methods, which have been inherited from prior proposals [85]. While these approaches have proven to be effective, current practices show significant lacks and room for improvement as discussed in both Part II and Part III.

The primary objective of this thesis is to optimize the sequence of steps involved in network traffic classification and introduce novel techniques to enhance key aspects in both the ML and networking domains.

## 1.3 Problem Statement and Contribution

To better formalize the question posed in the preceding motivation section, I split my problem statement into two research questions described below.

### 1.3.1 Highly Discriminant Feature Spaces

The first research question that I seek to answer in this thesis relates to feature engineering for NTA. Specifically, I aim to determine *whether it is possible to design discriminant feature vectors that improve NTA in terms of speed, accuracy, or explainability.* While addressing this primary question, I also consider several associated questions that arise in the context of NTA, such as the limitations or drawbacks of established feature vectors and the potential for improving them, the possibility of exploiting network traces in novel ways, and whether previous work has overlooked any critical aspects of NTA.

These questions are explored in the first part of the thesis, with the ultimate goal of proposing new network traffic representations, evaluating their performance, and comparing them to established solutions. The proposed representations are designed to address a range of tasks related to the use of ML for NTA, classification, and, most importantly, anomaly detection.

### 1.3.2 Enhanced Classification Frameworks

The second research question tackled in this thesis focuses on the tools, algorithms, and models employed for network traffic classification and anomaly detection. Specifically, I aim to determine *whether existing architectures can be enhanced by proposing novel concepts and refinements*. Given the broad scope of this question, the improvements discussed in this thesis revolves around enhancing the architectures by optimizing training and learning procedures, reducing inference time, and minimizing time complexity

In the second part of the thesis, I discuss these aspects in more detail and present my proposed approaches for improving them. The developed methods are evaluated and compared to equivalent approaches where appropriate.

### 1.3.3 Contribution

Based on the problem statement conveyed through the two research questions above and explored throughout this thesis, I summarize my contributions in the following:

#### Feature Spaces (Part II)

- The development of the *multikey* feature vector, a new multi-purpose, cross-layer network traffic feature space that is additionally able to cope with traffic encryption (Chapter 1).
- The development of I-Notice, a framework for sequence-based network packet representation with a compatible ML model for anomaly detection, which includes a prediction explainability study and continuous-learning capabilities (Chapter 2).
- The development of a representation and a compliant ML model that can estimate the number of multiplexed flows in network tunnels such as Virtual Private Networks (VPNs) (Chapter 3).
- Study, analysis, and comparison of a set of feature space scaling techniques as well as their implications on ML-based NTA (Chapter 4).

**Enhancing Detection and Classification Methods (Part III)**

- The proposal of Observer-based Data Modeling (ODM), a new algorithm for data sampling and low-density model extraction that is evaluated on improving and speeding-up training (Chapter 1).
- The conception of FlowGan, a framework for minority class augmentation and generation of network traffic at the flow and packet levels (Chapter 2).
- The proposal of EagerNet a neural network with an optimization allowing for faster predictions and less resource consumption (Chapter 3).

### 1.3.4 Published Material

- Fares Meghdouri, Félix Iglesias Vázquez, and Tanja Zseby. Modeling data with observers. *Intelligent Data Analysis*, 26(3):785–803, 2022
- Fares Meghdouri, Félix Iglesias Vázquez, and Tanja Zseby. Shedding light in the tunnel: Counting flows in encrypted network traffic. In *2021 International Conference on Data Mining Workshops (ICDMW)*, pages 798–804. IEEE, 2021
- Fares Meghdouri, Thomas Schmied, Thomas Gärtner, and Tanja Zseby. Controllable network data balancing with gans. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021
- Fares Meghdouri, Félix Iglesias Vázquez, and Tanja Zseby. Anomaly detection for mixed packet sequences. In *2020 IEEE 45th LCN Symposium on Emerging Topics in Networking (LCN Symposium)*, pages 120–130. IEEE, 2020
- Fares Meghdouri, Félix Iglesias Vázquez, and Tanja Zseby. Cross-layer profiling of encrypted network data for anomaly detection. In *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 469–478. IEEE, 2020
- Fares Meghdouri, Maximilian Bachl, and Tanja Zseby. Eagernet: Early predictions of neural networks for computationally efficient intrusion detection. In *2020 4th Cyber Security in Networking Conference (CSNet)*, pages 1–7. IEEE, 2020
- Maximilian Bachl, Fares Meghdouri, Joachim Fabini, and Tanja Zseby. Sparseids: Learning packet sampling with reinforcement learning. In *2020 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9. IEEE, 2020
- Félix Iglesias, Fares Meghdouri, Robert Annessi, and Tanja Zseby. Ccgen: Injecting covert channels into network traffic. *Security and Communication Networks*, 2022, 2022

CHAPTER 2

# Background

According to recent reports, cyber-attacks have increased remarkably in the last few years [201, 39], implying a greater need for secure infrastructures than ever before. Fortunately, experts and researchers have invested much time and effort into developing robust methods for defending against modern threats. Side by side with traditional methods, Artificial Intelligence (AI) and, more specifically, ML has recently been deployed in communications networks to aid in attack detection. Such approaches have proven to perform satisfactorily and assist in solving problems related to modern challenges, such as anomaly detection with the increasing proportion of encrypted traffic that hinders traditional DPI techniques [200] and the extensive resource requirements or slow reaction time observed in highly complex and dedicated Intrusion Detection Systems (IDSs) [216].

Nonetheless, this rise of applied ML in the networking and cybersecurity domains necessitates the practical adaptation of existing theoretical solutions. Although certain architectures can be easily transferred across domains (particularly those that serve the same goal like classification), careful consideration must be given to several factors, particularly those related to input data especially when dealing with feature engineering. Neglecting theoretical aspects such as data characteristics can compromise the success of ML algorithms, and evaluating performance solely based on performance scores may overlook potential vulnerabilities, even with the use of powerful models such as (deep) neural networks. This problem becomes extremely relevant in NTA due to its importance in today's infrastructure.

In this chapter, I establish a foundational basis for all proposals in the thesis. An overview of the design and implementation of most ML-based models for NTA is provided in what follows, outlining each component and examining algorithms and models, along with a few adaptations and frameworks inspired by other similar domains.

## 2.1 The Classical ML-based Framework for NTA

When designing a solution for NTA that is based on ML, a conventional architecture is mostly used. Specifically, the set of components performing various functions, from data acquisition to model inference have undergone some internal changes over time, but the fundamental structure of the architecture remains largely unchanged. Figure 2.1 shows an overview of these components and their interconnections; their seamless integration facilitates the optimal utilization of ML algorithms with network traffic data, resulting in efficient and effective NTA systems.

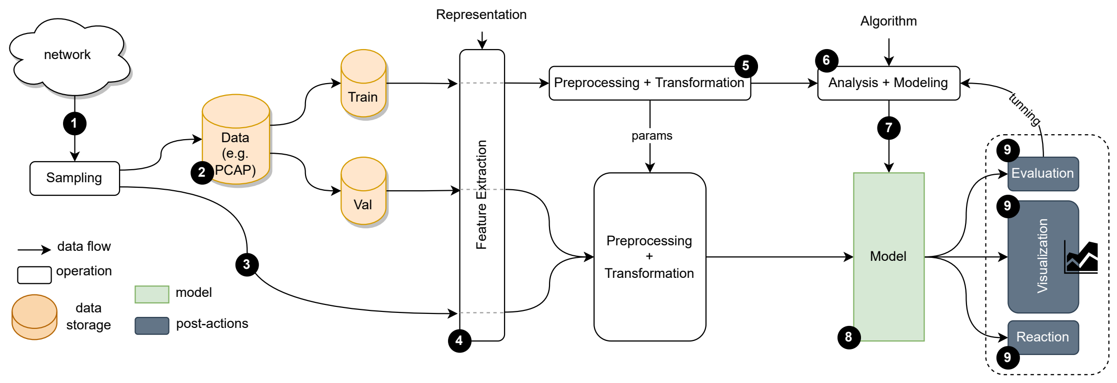


**Figure 2.1:** The Classical ML-based Framework for NTA.

In what follows, I describe each component in detail.

### 2.1.1 Capturing Network Traffic

❶ - There exist several topologies that connect network devices. These topologies can be established through different physical mediums that are generally straightforward to access. This accessibility facilitates the capture of device-to-device traffic for various purposes, ranging from benign to malicious. Traffic can be recorded either at one point or simultaneously at multiple locations. If necessary, this traffic can then be aggregated using highly granular time synchronization across sniffing devices.

Two leading technologies that enable the (active or passive) collection of network traffic:

**Test Access Points (TAPs)** are full duplex passive devices that duplicate the same data in both directions of a link and provide the option to mirror it to a third device (typically a storage or fast monitoring hardware). They ensure that the traffic has not been altered in any way.

**Switch Port ANalyzers (SPANs)** are a technology rather than a device that allow traffic collection; they are represented by physical ports on various network devices such as switches and routers. Typically, they can be programmed to mirror/forward specific network packets. As a result, they can induce network delays and/or modify packets in a controlled manner. These devices are often not intended for fast backbone links with exceptionally high throughput.

### 2.1.2 Collection and Storage of Network Traffic

❷ - In most NTA designs, portions of the collected network traffic are saved and utilized for various purposes, including training, validation, and analysis of ML models. As a result, the first component after a network capturing device is typically a storage unit designed to handle vast amounts of data. Network traces are commonly stored in the form of Packet CAPtures (PCAPs), which adhere to a well-defined standard and enable for universal usage and compatibility with other Operating Systems (OSs).

To preserve privacy or save storage space, it is common practice to truncate packets in a network capture to a specified length or entirely remove the payload if it includes sensitive or unusable encrypted information. Additionally, sampling may be employed to store only a portion of the packets.

### 2.1.3 Online Inference

❸ - Another technique for handling network traffic is to process it online, which is used when designing systems that are already preconfigured and require immediate action. This approach is frequently used for online inference after developing and validating a model using the data in stage ❷. In this scenario, packets are usually buffered in a First-In First-Out (FIFO) queue and released continuously. To avoid bottlenecks and overflow, the processing bandwidth of the subsequent components should be similar to the link's capacity. Sampling can be utilized to select only a portion of network packets if the hardware is less powerful.

### 2.1.4 Feature Engineering and the Feature Extractor

❹ - After collecting the data, the subsequent step involves creating a feature space by extracting discriminant attributes that accurately depict the network state. This process is commonly known as feature engineering, which encompasses the transformation of raw data into a distinct representation. In the context of network traffic, the challenge lies in converting the collection of packets, or PCAPs, recorded at a specific network point into a representative set of numerical values. Current techniques involve aggregating network flows using the flow-key, computing statistics and interactions among packets, and extracting protocol-based flags and payloads. The resulting data is represented as a feature vector that is utilized for further analysis. Packet sampling can also be applied at the flow level to enhance training and speed up inference [64].

In the following, a definition of a network flow is provided and will be utilized throughout the thesis. Modifications to this definition, if introduced, will also be explicitly stated in their respective sections. A network flow is defined as a collection of packets observed on the network with common properties, as specified in [58]. At the fundamental level, these properties include the source and destination hosts, which are respectively represented by their Internet Protocol (IP) addresses. Furthermore, unique identifiers such as source and destination transport ports and the network protocol, combined into a "flow key", can be used to identify a network flow. This additional information enables the identification of the application responsible for the transmission. The collection of packets, or flow, is typically treated as a sample and passed to the subsequent components as described earlier. Parts of this thesis investigates whether fewer or more identifiers are preferable in a flow key. It is important to note that several extraction tools have been developed to convert raw PCAPs into structured data. In this work, I mostly use *go-flows* [241], an open-source, highly customizable flow extractor. Furthermore, the authors of this tool discuss how flow extractors can significantly affect the resulting data and how this effect must be considered in the subsequent analysis [241].

### 2.1.5 Preprocessing

❺ - After collecting and aggregating features into vectors, the data must undergo a preprocessing step to transform it into an appropriate type, distribution, etc., for compatibility with the algorithm(s) used later. Depending on the situation, processing may involve various procedures, e.g., changing ranges and distributions (scaling and normalizing), converting categorical attributes to numerical ones (e.g., One-Hot-Encoding (OHE)), and space transformation (e.g., Principal Component Analysis (PCA)). However, processing data may result in information loss, such as when using PCA for feature reduction. Fortunately, the data loss may not always be significant and can even be beneficial for noisy data in some practical scenarios.

### 2.1.6 Analysis and Modeling

❻ - The fundamental step of any ML-based approach for NTA involves analysis and modeling. This process may involve qualitative analysis utilizing statistical tools and methodologies or building a mathematical representation (i.e., model) with previously processed data. Analysis encompasses all the steps required to comprehend the data, its shape, and characteristics. On the other hand, modeling aims to establish a model of the data (e.g., in Unsupervised Learning (UL)) or a connection between the observed data and a target value, typically referred to as a label, probability, or prediction (e.g., in Supervised Learning (SL)). The obtained "model" is then used to address the problem with future observations. These models differ significantly from algorithm to algorithm and may be a collection of parameters, rules, or probabilities. Further elaboration on this topic is presented in each chapter of the thesis.

### 2.1.7 Model Extraction and Deployment

❼ - Models are created, stored, and deployed to generate predictions for future data. As discussed in the preceding section, a model is essentially a compilation of parameters and rules organized according to a predetermined structure, such as the coefficients of a polynomial equation. The prediction (i.e., target value) is determined by combining these coefficients mathematically with the input vector (e.g., element-wise multiplication). To validate a model, a framework is evaluated through specific testing methods based on the application to ensure accuracy, explainability, and adversarial robustness. Other techniques, such as federated learning, can also be employed to build models. Once a model is generated, it is deployed for a defined period before being replaced to address potential data drift.

### 2.1.8 Real-Time Evaluation

❽ - Trained models are often exported for live system deployment; they can be spread over numerous nodes or utilized on a single server. The deployed model is often built to cope with streaming or static data and provides output values in real-time, regardless of whether it is a network monitoring system or a Network Intrusion Detection System (NIDS). The feature extraction and processing stages remain the same and are used similarly to testing. It is worth noting that inference speed is critical and relates to network bandwidth; consequently, sampling or other optimization techniques are typically used.

### 2.1.9 Decision Making, Evaluation and Visualization

❾ - After tuning all the internal parameters of a particular model (step ❻), an output value, prediction, or probability is generated by assessing the input vector. Based on the sensitivity of the application, an action may be executed automatically, such as blocking access from a specific IP address, or manually for more severe cases, such as shutting down the entire company's network. In critical networking infrastructures, such as those connected to the Internet backbone, automated actions are less preferable **unless** they meet specific requirements, such as a specific automation rate or False Positive Rate (FPR). Instead, experts are typically responsible for receiving prompt alarms, investigating the situation, and deciding on subsequent actions. Visualization techniques, such as histograms or time series, can be utilized to examine various aspects of the evaluation that may be difficult to comprehend with just numerical data. Furthermore, analyzing data clusters or monitoring data and model drift over time may provide insights into the current state of the system.

## 2.2 Applications and Algorithms

Depending on the nature of the problem undergoing analysis and the task in question, ML algorithms can be classified into three categories. In the following sections, I outline the majority of algorithms employed throughout the thesis and provide generic examples of algorithms and strategies for solving particular problems. Most algorithms are mathematically well established; I recommend that readers interested in the mathematical foundation consult [96].

### 2.2.1 Supervised Learning

Supervised learning refers to algorithms that are meant to learn an objective function from a set of data and a set of desired values. Let $\mathbf{x}$ be a feature vector obtained in the way described in Section 2.1.4. Let $\mathbf{y}$ be a target value that corresponds to $\mathbf{x}$; for simplicity, we suppose that in anomaly detection $\mathbf{x}$ is a network flow representation and $\mathbf{y}$ can take two values: 0 and 1, denoting a benign flow and a malicious flow, respectively. The vector set $\{\mathbf{x}_0, ..., \mathbf{x}_n\}$ is in the input space $X$, and the scalar $y$ is in the output space $Y$. $y$ can also be a vector in the case of multi-label problems ($\{\mathbf{y}_0, ..., \mathbf{y}_n\}$). As previously stated, there is a mapping between both sets, and so the dataset may be expressed as the pairs $\{(\mathbf{x}_0, \mathbf{y}_0), ..., (\mathbf{x}_n, \mathbf{y}_n)\}$. In supervised learning, the algorithm finds a mapping function $f$, from a collection of functions $F$ such that the mapping $f : X \rightarrow Y$ is approximated as good as possible. It is worth noting that the task of finding $f$ can be solved using a variety of algorithms, and the input data structure can be much more complex than a simple one dimensional input vector. Some algorithms, for example, can deal with structured data, unstructured data (such as images), and sequences (such as time series). Recurrent Neural Networks (RNNs), LSTMs, and Convolutional Neural Networks (CNNs) are some examples of those algorithms.

#### Classification

The target value is regarded as a discrete class label in classification. In this scenario, the purpose of the model is to predict the class of a given input sample $\mathbf{x}$ from a list of predetermined classes. In the context of this thesis, the class may be a traffic category, a network service, or a family of attacks. The target variable might be one-dimensional or multidimensional, representing a discrete variable, binary vector (for example, when using One-Hot-Encoding (OHE)) or a set of labels. Some SL classification algorithms are: Decision Trees (DTs), Random Forests (RFs), Naive Bayes (NB), k-Nearest Neighbors (k-NN), and so on.

#### Regression

When it comes to regression, the target variable or vector is not discrete but rather continuous, implying that the output is a sort of combination of the inputs. Based on the input vector, the aim is to derive a value from a continuous space. In this thesis, I use regression for instance to estimate the number of packets. It should be noted that a regression-based approach may also be utilized to predict descrete values e.g., by applying a threshold to the output. Regression techniques include linear regression, logistic regression, and, eventually more advanced techniques such as Fully Connected Neural Networks (FCNNs) and Support Vector Machine (SVM), among others.

Semi-SL is a category similar to SL. In this case, labels from only a few categories are available, and the rest are marked as unknown. This is especially useful in NTA when a large variety of labeled normal network traffic is knows a-priori and considering the rest as abnormal, it can be thus particularly good at detecting zero-day attacks.

### 2.2.2 Unsupervised Learning

Unsupervised learning refers to learning from data that does not include a target variable, like with SL. Instead, the algorithms in this category learn probability distribution functions and similarities between input samples rather than a relationship between input and output. Although not always true, typically the larger the number of samples available, the better generated models reflect insights about the structure of the data and the various patterns within it. Unlabeled data is a term researchers use to describe data incorporated for UL. There are a variety of approaches for learning from unlabeled data, as well as a variety of applications. Some of the use-cases are discussed in what follows.

#### Clustering

Clustering is one of the most well-known ML application; it represents the task of grouping data samples that show similarities to each other into a cluster. A cluster is typically described as a group of points with similar properties. Clustering algorithms establish similarities based on different measurements and distances and is usually utilized in NTA to discover similar network services, bottlenecks, or anomalies (which are samples with very few similar instances).

#### Dimensionality reduction

Through the process of learning data structures, it is possible to reduce the number of dimensions in the input space while retaining only the necessary information for approximately the same level of variance. Algorithms within this category are commonly employed to create a data model, which then translates the original space into a subspace with fewer dimensions while preserving nearly the same amount of information (variance). This is especially important in applications where the input space is exceedingly large, or data visualization is required but the space contains more than three dimensions. Utilizing color and size, can be implemented to visualize up to five dimensions and other tricks can be employed to go beyond. In the context of NTA, dimensionality reduction is of particular importance for visualization purposes and more broadly for eliminating low variance or noisy network features. This, in turn, improves subsequent analysis components, including training and producing faster predictions.

#### Association

Rule association, like the previous two applications, is another application derived from understanding the data structure. An association algorithm attempts to discover correlations between attributes of a given sample. This enables the resolution of a diverse range of problems without the need for labels. At its essence, association may be defined as follows: For a given vector $\mathbf{x}_i$, the purpose is to determine the most likely $x_{i,j}$ given a collection of other vector values $x_{i,m}$ for $n = 0, ..., n - \{j\}$. This may be used in NTA to mitigate the effect of traffic encryption, for example, by filling in missing values.

Nevertheless, in some cases, association rules might also be considered as SL, specifically while implementing the training procedure i.e., by taking one of the variables as label.

### 2.2.3 Reinforcement Learning

Reinforcement Learning (RL) is yet again an important component of ML paradigms. RL algorithms enable action-making based on feedback from a system state. The goal here is to identify the optimum set of actions to perform in order to attain a particular goal. RL, like UL, does not require data labels but instead optimizes based on a "reward" acquired each time the model selects an action. Typically, RL models take a state vector $\mathbf{s}$ and a reward value $r$ as inputs and output an action vector $\mathbf{a}$. The models are commonly referred to as agents or actors, and their objective during training is to learn the optimum action vector $\mathbf{a}$ given the prior inputs $\mathbf{s}$ and $r$. Many architectures have been developed to address this topic, demonstrating astounding performance across various areas. In NTA, RL has been used for queue management, dynamic link capacity control, and dynamic packet sampling. Algorithms employed include Q-learning, AC learning, policy optimization, and others.

## 2.3 Adaptations From Other Domains

Many of the proposed architectures and models used in NTA are inspired by other domains and the success achieved in resolving various issues. In what follows, I discuss two popular domains on which many of the used models here showed great success.

### 2.3.1 Computer Vision Applications

In Computer Vision (CV), researchers typically use ML to solve tasks involving a machine's ability to observe and understand the real world. Object recognition in images is a subtask typically handled by ML algorithms. The CNN architecture is one of the most recent architectures demonstrating outstanding performance in image classification. Internally, the architecture employs convolution filters to learn contextual features from images and relate them to object detection. Because of their ability to detect patterns, CNNs (specifically uni-dimensional networks) can and have been used to some extent for NTA. Because the convolutional part of CNNs symbolizes a feature extractor, the same concept can be applied to network traffic to create a feature extractor that captures relevant information from packets, for example.

A further flourished architecture are generative models, especially Generative Adversarial Networks (GANs). Many configurations and optimizations have been proposed to improve their quality. GANs are widely used in CV to generate images and videos. In NTA, GANs are primarily used to generate network traffic by mimicking previously observed data, whether benign or malicious.

Another generic-inspired architecture is the Auto-Encoder (AE), which is a neural network optimized to have an output similar to the input thus, it can learn to reconstruct the same information. It is often used for compression due to it's triangular shape but also can be used with images, for example, for denoising. It has been widely used for NTA and anomaly detection based on the reconstruction error, which tells whether the output deviates significantly from the input indicating a possible anomaly, i.e., a sample that has not previously been observed during training.

### 2.3.2 Natural Language Processing (NLP) Applications

NLP has recently achieved remarkable success, which can be attributed to the availability of data and the support of well-known research organizations such as OpenAI[1] (e.g., GPT, chatGPT, Whisper). The primary objective of NLP is to comprehend and generate language expressions, such as speeches, sentences, and the context of a paragraph. Recurrent models have been widely employed in this regard, particularly with the abundance of data. Models like RNNs, their successor LSTM networks, and Transformers have shown outstanding results in these applications. This is because their architecture can handle sequences of words and phrases, and their internal structure introduces the attention principle, allowing for memorization of the most important elements in a sequence. Recently, Transformers have been proposed, which are similar to LSTMs but use the attention mechanism to achieve even more realistic text generation. Although probabilistic and likelihood-based models, such as Markov Chains, have been effective for the same task, deep learning has proven to be superior. Recurrent and sequence-based architectures are also widely used in NTA, as network traffic can be viewed as time series, fitting the described scenarios perfectly. Previous NLP models can be easily adapted and employed to solve tasks such as classification, anomaly detection, or traffic generation.

[1] `openai.com`

CHAPTER 3

# Challenges and Requirements

Some requirements must be met when developing a new system, and NTA and anomaly detection frameworks are no exception. In fact, using ML to solve a wide range of problems results in stricter requirements. In this section, I go over the main key points to consider when creating a ML-based framework for network traffic.

## 3.1 Type I and Type II Errors

When dealing with anomaly detection (or network traffic classification in general), detection accuracy is a crucial evaluation metric to consider, which represents the correctness of the predictions on average. There are many other metrics for evaluation, but they are retrofitted to deal with data imbalance, amplify specific classification behavior, and so on. All of these metrics are typically derived from a confusion matrix, which in the case of a binary classification (e.g., predicting attack/non-attack), contains four values related to two hypotheses described in the following. To simplify matters, a hypothesis in statistics is the suggested value of a variable which can be confirmed or denied following a test that can be, for example, an observation. The hypothesis can be then accepted or rejected after the observation. If the null hypothesis is true in the case of anomaly detection, it means that the observation (in this case, a network flow) is not an attack. On the other hand, a false null hypothesis indicates that an observed flow is an actual attack. The test results can then be summarized in the confusion matrix by including all four cases: true null hypothesis with an accept, true null hypothesis with a reject, false null hypothesis with an accept, and false null hypothesis with a reject. Each of the cases has a specific meaning; when the type and decision on the hypothesis match, we have a correct decision or prediction (e.g., an attack is correctly detected as an attack). When we do not have a match in the remaining two cases, it means we made one of two mistakes: rejecting a true null hypothesis (type I error) or accepting a false null hypothesis (type II error) [24]. These two errors are commonly referred to as false positive and false negative in practice. The former represents raising a false alarm about a normal network flow, and the latter represents failing to detect a malicious network flow. To summarize:

- Type I error: False Positives (FPs), predicting normal network flows as attacks.
- Type II error. False Negatives (FNs), predicting attacks as normal network flows.

In statistics, probability density functions about the null- and alternative hypotheses are typically provided or estimated. Thus a threshold $\alpha$ can be manually selected to account for the number of FPs and FNs. However, decreasing one value might raise the second, causing a trade-off. The trade-off in ML, and precisely when training models that provide probabilities at the output, can be established by setting a threshold value, which is typically 0.5, meaning that a probability estimation smaller than 0.5 is converted into the 0 class and a probability value greater than 0.5 is converted into the 1 class.

Ideally, we aim to minimize both types of errors. For example, in type I: raising too many alarms may introduce overhead that is not desired, such as manual inspection from domain experts and access denials when an automated system is implemented. With type II, network attacks may go unnoticed, defeating the purpose of an IDS. However, there has always been a debate about which error types should be reduced or prioritized [59]. This is still an open question, but rather than deciding where to set the threshold, one should aim to build a model that performs well and minimizes both.

Another point to consider is the metrics used to evaluate trained models. FPs and FNs are typically not provided straightforwardly, but rather a single metric is used to reflect the performance of a model, such as accuracy. However, some metrics defined in the literature may not accurately reflect both error types. For example, the accuracy does not tell us much about how many FPs or FNs there are or their proportions, so we have no idea about a model's sensitivity. To address this issue, researchers typically use other specific metrics to show whether a model generates more FPs, FNs, or both. FPR (Rate), FNR (Rate), Precision, and Recall are examples of such metrics. Other adjusted metrics can be subsequently derived from these to solve problems related to data imbalance, in the case of anomaly detection, for instance.

### 3.1.1 Metrics Used Throughout the Thesis

In this section, I summarize all metrics used in this work. Since I am mostly dealing with classification problems, a confusion matrix is typically derived from results and four quantities (as explained in Section 3.1) are computed, namely FPs, FNs and:

- True Positives (TPs), predicting attacks as attack network flows.
- True Negatives (TNs), predicting normal network flows as normal.

From these four quantities further metrics can be derived, such as:

$$Accuracy = \frac{TP + TN}{TP + +TN + FP + FN} \tag{3.1.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{3.1.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.1.3}$$

From Precision and Recall, further composed metrics can be obtained from the Precision-Recall Curve which is a curve obtained by computing the previous metrics for various threshold values. Some derived metrics are:

- Average Precision: which is defined as the area under the PR Curve divided by the total number of relevant items.
- Weighted Average Precision: similar to the previous one but considering the class weights (micro-average) for the averaging.
- Weighted Recall: can be computed without the PR Curve. It is obtained by computing the recall for each class in the dataset and then averaging using class weights (micro-average).

Using macro and micro average, further versions of the above metrics can also be obtained, i.e., in the case of macro metrics, the average is done without considering the class distribution in contrary to the micro metrics where the average is weighted.

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{3.1.4}$$

$$MacroF1 = \frac{2 \times MacroPrecision \times MacroRecall}{MacroPrecision + MacroRecall} \tag{3.1.5}$$

Similar to the previous weights metrics, a weighted F1 score can be obtained by computing the F1 score for each class and then averaging the result using class weights.

$$Youden'sJ = Recall + \frac{TN}{TN + FP} - 1 \tag{3.1.6}$$

Another metric is the the ROC-AUC which is calculated by plotting the True Positive Rate (TPR) against the FPR for different classification thresholds and calculating the area under the curve.

The aforementioned metrics are crucial for evaluating the effectiveness of a classification solution, particularly in situations where class imbalance may skew the results. Some are also necessary for various applications, including the evaluation of error measures, clustering validity, and outlierness scores. Further elaboration on these metrics can be found in their respective chapters.

## 3.2 Prompt Detection and Response

Increasing data volumes exchanged over the Internet necessitates the deployment of faster links to interconnect components, particularly in backbones [11]. This has been the trend over the years and implies that network devices must be capable of handling increasing bandwidth and large data amounts. Network devices used to capture packets, such as TAPs and SPANs, have also been improved to handle such speeds. However, the challenge in anomaly detection and NTA remains in the following components, as mentioned in Figure 2.1. In general, and regardless of the number of components used, two requirements are critical for successfully deploying a ML-based networking system: fast detection and fast response.

Presuming that developed systems do not have infinite computational resources, optimization will always be required to handle all packets observed in the network, including parsing, storing, processing, and executing computation logic which becomes a challenge, especially in Gigabit links. On the other side, network attacks, in particular, must be detected quickly, and automated or manual responses must be implemented in real-time to mitigate the damage.

Many proposals have been developed to improve the detection speed of ML-based systems, particularly in the networking domain [230, 194, 235, 212, 190]. Algorithms, for example, have been significantly optimized to reduce computation complexity. Other solutions include 1) sampling packets over time, which has been proven to perform plausibly, 2) optimizing the executed code by avoiding high-level programming languages focusing more on low-level that interact directly with the hardware and thus all interfaces, including the network interface, 3) the use of more powerful hardware is also considered a solution, but it is not always feasible. In Internet of Thing (IoT) infrastructures, for instance, a lightweight version of an IDS (e.g., a neural network with less layers) is often implemented and deployed on end devices with limited resources [61], so the first two options may be better in this case.

On the other hand, automated systems and experts should respond quickly to alarms from detection components to limit the damage or spread of malware or attack, particularly in the case of zero-day attacks [32].

In the development of any NTA system, it is crucial to optimize for speed and carefully consider the bandwidth of the link under observation to avoid creating a bottleneck. Accordingly, in this thesis, the evaluation of this aspect is performed, and the time complexity of each proposed solution is optimized.

## 3.3 Memory and Storage

When developing ML schemes, storage is a crucial requirement that must be addressed, particularly in the context of NTA. Memory and storage requirements are influenced by several factors, such as the need to store significant datasets for training, validation, and testing purposes [8, 10]. Additionally, large models, which are typically represented in the form of graphs, weights, rules, or probabilities, need to be stored. Lastly, in online implementations, samples must be stored for incremental learning, where the original dataset is no longer needed, but newer samples are continuously collected. All of these factors must be carefully considered when optimizing the storage requirements of a NTA system.

A few methods can address these issues, such as delegating the task to a storage unit that handles all operations. However, in some applications, such as federated learning [141], data and models are distributed across multiple nodes, necessitating more extensive individual storage capabilities. In this case, cloud storage can be used but must be accessible by all network devices. This alleviates requirements, but only if the access is reliable and stable. A different approach is to use data subsets that include the most recent samples while discarding previous ones. Part II - Chapter 2 and Part III - Chapter 1, I show that this is possible and produces comparable results to initially storing the entire data set.

## 3.4 Traffic Encryption

***Notice of adoption from previous publications - 1***
*Parts of this section's content have been published in the following paper:*
*Fares Meghdouri, Félix Iglesias Vázquez, and Tanja Zseby. Cross-layer profiling of encrypted network data for anomaly detection. In* 2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 469–478. IEEE, 2020*

In recent years, traffic encryption has become a significant challenge for NTA in general. Although encryption protects users' privacy, it also makes it difficult to develop network monitoring systems. After traffic is encrypted, little information is left to explore, making it difficult to extract discriminative information. In this thesis, I frequently investigate the impact of two secure communication protocols (Transport Layer Security (TLS) and IP Security (IPsec)) on proposed detection methods. These protocols are widely deployed and are briefly discussed below.

### 3.4.1 TLS

TLS [197] is a secure communication protocol that provides confidentiality and data integrity on top of the transport layer. For instance, it is a commonly used protocol with Hypertext Transfer Protocol Secure (HTTPS) to offer secure web browsing. During the TLS handshake, a secure connection is established, after which encrypted data is transmitted.

Since TLS is running on top of the transport layer, all fields of the IP and TCP headers remain unencrypted. Furthermore, TLS encrypts all packets of a single TCP connection individually without tunneling; hence, connections can be analyzed separately. In TLS 1.2, some parts of the initial handshake are transmitted unencrypted (e.g., signature exchange for mutual authentication), which allows further analysis during the connection establishment. However, in TLS 1.3, several steps of the initial handshake are already encrypted, reducing analysis possibilities. The Datagram Transport Layer Security (DTLS) protocol [198] is a variant of TLS that provides security features for datagram protocols (such as User Datagram Protocol (UDP)). The encrypted packets look similar to the TLS packets, but DTLS still needs to deal with packet loss and reordering by adding sequence numbers at the application layer.

### 3.4.2 IPsec

IPsec [86] is an end-to-end security protocol at the network layer. Security Associations (SAs) are established using the Internet Key Exchange (IKE) protocol. IPsec operates in different modes: transport mode and tunnel mode.

In transport mode, the endpoints of the security association are hosts. IPsec adds an Encapsulating Security Payload (ESP) header before the transport header, and everything after that is encrypted. This means that transport header fields cannot be used for traffic analysis. In IPv6, the ESP is implemented as an extension header, encrypting all subsequent headers. In addition, IPsec usually aggregates all traffic from one endpoint into one SA. However, it is possible to select which packets are encrypted at the endpoint, but typically all traffic from the source of the SA to the destination of the SA is encrypted, therefore aggregating multiple application flows from a host or a router. This aggregation of several application flows into a larger *source-destination* flow further impairs analysis possibilities.

In tunnel mode, the IPsec SA is established between two routers, meaning that traffic from the hosts to the router remains unencrypted (e.g., because the local networks are trusted). The router later establishes a tunnel and encrypts all the traffic. Here, the original IP header (with the IP addresses of the source and destination host) is also encrypted, and a new IP header with the IP addresses of the router endpoints is added instead. In this case, the original host IP addresses cannot be recovered from the encrypted traffic. Since multiple hosts can use the tunnel, the level of aggregation is even higher than in transport mode. The tunnel mode also works with a mixed setup with a host and a router as endpoints. One specific example is the establishment of a VPN. In such a case, the SA starts at a host (which, for instance, visits an untrusted foreign network) and ends at a gateway (which is the entry point to the trusted home network). In this case, the new header contains the original host IP address as source IP address and the entry router of the home network as destination IP address.

Throughout the thesis (especially Part II), the impact of securing the communication is investigated and solutions to solve problems induced by encryption, such as the inaccessibility to features as well as how to overcome this issue are discussed.

## 3.5 Privacy Constraints

Regarding privacy, regulations governing which data is legally permitted to be collected have become more stringent in recent years, particularly since the implementation of General Data Protection Regulation (GDPR) compliance. However, when talking about statistical NTA, user-related private data are typically not used but only features related to their behavioral profile, which includes statistics about the interactions a user generates while browsing a web page, downloading a file, or sending an email. As a result, the actual content (i.e., DPI) is not used - especially in the context of this thesis. However, legal instructions generally prohibit also this kind of data collection, posing a significant challenge for network experts [182]. On the other hand, it is evident that large amounts of data are required for successful ML applications for better training; thus, limiting the data that can be collected results in sub-optimal training. Anonymizing the data by swapping identifiers or content that refers directly to users while maintaining the data structure is one way to overcome this. However, some approaches still have a shortcoming in which users can be referred to even after anonymization, particularly when many identifiers have a high degree of uniqueness. This problem is addressed in the literature under k-anonymity [210].

One of the solutions to overcome privacy constraints while still collecting enough samples is to artificially generate network traffic. So far, the following methods have been used: 1) replicating user behavior with simulation tools in a controlled environment [5]. Thus, there are no constraints, but the traffic does not entirely reflect reality, yet it is a good place to start. Typically, user profiles are extracted from real traces and then modified for reproducibility. 2) Another recent development uses generative networks, such as GANs, to generate artificial data from initially collected sample sets [203, 53, 218, 50, 154]. The difference between the two approaches is that the first allows for generating accurate, valid network traffic, whereas the second one generates samples that have continuous distributions, which may result in invalid values in some fields, such as floating protocol numbers. Other hybrid approaches also exist in which artificial traffic is mixed with real one to construct the entire dataset however, artificial network traffic generation still has shortcomings regardless of the methods used (see next section). Unfortunately, it is sometimes the only way to overcome constraints imposed by privacy regulations for the time being.

## 3.6 Data Artifacts and Technical Issues

Experts need to ensure that the network traffic data used for training is free of artifacts that could significantly deviate the data from its theoretical distribution. Such artifacts can adversely impact the choice of transformation techniques, leading to a reduction in the quality, robustness, and generalization of the trained predictor against novel traffic. In the following, I highlight three prominent artifacts I observed during the analysis of several publicly available intrusion detection datasets. Further issues are discussed in [155]. Although these artifacts can affect the subsequent analysis, they are often unavoidable since they are inherent to the data generation process and typically require a distinct approach to mitigate them.

### Uni-source traffic generation

During the analysis of the UNSW-NB15 dataset, an intriguing behavioral pattern was discovered, particularly with regards to the Time-to-Live (TTL) feature and its derivatives. This feature exhibited a significant contribution to the decision-making process for detecting malicious flows. However, further investigation revealed that this was primarily due to the dataset's generation process. The dataset's authors generate attacks in a supervised simulation environment, utilizing a single host to produce various types of attacks across various time windows. Since packets originated from the same host, all malicious attacks shared a similar TTL value or range. Consequently, the classifier blindly learned this relationship, resulting in the decision-making process relying solely on the observed TTL value.

### Packet order

Packet reordering, caused by packets taking different network routes, can result in packets arriving at the receiver in a different order, although this occurrence is infrequent. Such an event can be detected through the use of time-related features, such as packet Inter-Arrival Time (IAT), in the context of NTA. When packets are reordered to form meaningful flows (e.g., in TLS), the IAT may become negative, thereby altering the data distribution. Contrary to intuition, the time-delta is always positive, and as such, caution must be exercised when applying transformations that assume strictly positive values during preprocessing, such as the log or power transform. In this thesis I henceforth raise awareness of the negative time-delta phenomenon resulting from the behavior of communication networks. Therefore, it is necessary to be cautious when applying transformations that assume positive values to such features [254].

### Duplicate packets

Although duplicate packets were not observed in the datasets analyzed in this work, [118] reported an instance of such an occurrence in the MAWI dataset. These duplicates were caused by misconfigurations in the capturing devices. Duplicate packets can alter the observed behavior of network traffic, leading to changes in the distribution of individual features and generating incorrect statistics, ultimately resulting in erroneous decisions. It is worth noting that the UNSW-NB15 dataset also encountered this issue due to capturing traffic on both ingress and egress interfaces of the TAP device. However, the current release (Jan. 2023) no longer suffers from this issue.

## 3.7 Explainability and Trust

Finding the reasoning behind predictions obtained from a model is one of the most critical aspects of implementing abstract ML models. By explainability, we mean understanding how the method fits by knowing which input values contributed to a particular output. In fact, for a long time, many ML models were regarded as black boxes that did not provide an explanation on why an output was produced, whether it was correct or not, but this started to change now [108]. Although most ML models are interpolative, which means that output values are limited to a range observed during training, explaining why a specific output value (also known as prediction) occurs remains obscure. Furthermore, some models deviate from this behavior and can be extrapolative (e.g., FCNN), with unexpected outputs for input vectors that were not provided during training, necessitating regulation before any actual deployment [26, 250]. Fortunately, many explainability methods (e.g., SHapley Additive exPlanations (SHAP), Local Interpretable Model-agnostic Explanations (LIME), Accumulated Local Effect (ALE), etc.) have been proposed in recent years; most of these methods are based on perturbing the input space and observing the effect on the output to develop an idea of which features are more relevant. Other ML algorithms (for example, DTs) provide feature importance, a metric that reflects how much a given feature contributes to the decision on average.

Explainability greatly impacts the viability of a proposed IDS because a well-established and easy-to-interpret model delivers more trust. In contrast, a black-box model introduces uncertainty and doubt [207]. Furthermore, models are preferred to react automatically in the event of a network attack, as discussed in Section 3.2; however, when trust uncertainties are introduced, this becomes difficult, and manual actions are typically preferred if models are not trusted. These trust uncertainties cannot be eliminated unless the developed model provides explicit explanations for each output value, which remains challenging [73]. Experts typically do not trust unpredictable models, and in some cases, decisions such as blocking access for a specific IP address cannot be made without implicit reasons. Therefore, such explanations are critical for all ML-based systems developed for intrusion detection and NTA in general.

Explainability analysis can also aid in discovering artifacts in the training set as discussed in Section 3.6. Artifacts in data sets can cause the model to perform reasonably well with most samples but fail with others but also a more severe issue is that the same model may still perform well and output the correct class in some cases, but for the wrong relationship. Investigating explainability is thus a favorable solution to find unreasonable explanations similar to the TTL example discussed in Section 3.6; and how a classifier correlated the input features to the label, yielding correct predictions but for the incorrect mapping function.

In this thesis, I conduct explainability analysis after each experiment and discuss the findings to dispel doubts about proposed models and frameworks.

## 3.8 Further Challenges

Depending on the application, there may be additional challenges such as detection speed, dealing with massive amounts of streaming data, obtaining labeled data or labeling network data, selecting the right features, studying feature correlations, and so on. However, I concentrated solely on the aspects pertinent to this work in the preceding sections. Nonetheless, a large scientific community is delving into each of the aforementioned points in depth.

CHAPTER 4

# Related Work

***Notice of adoption from previous publications - 2***
*Parts of this chapter's content have been published in the papers referenced in Section 1.3.4*

Enormous efforts and limitless attempts have been made to move to more stable, fluid, and dexterous systems from conventional static and rule-based classification and detection technologies for network traffic. Researchers have incorporated expertise from various fields and well-established functional methodologies on network traffic. In order to improve state-or-the-art performance obtained by using previous techniques, a trend focusing specifically on ML algorithms and their hybrid combinations with different preprocessing techniques began to develop.

ML by itself has been widely used in the last decade for network traffic analysis and, specifically, anomaly detection systems. Many of the works proposed IDS architectures based on well-known supervised techniques e.g., [41, 31]. The concept of having a classifier trained on pre-stored attack patterns and using it to predict similar behaviors is commonly used whereby remarkable success has so far been achieved, e.g., by [236, 84, 13]. Among others, RFs and FCNNs are commonly utilized whenever large network traffic datasets are available. In order to present network data to such architectures, several traffic representations have been proposed depending on the application. The statistical representation of network flows remains among the most popular.

All the above and related work has achieved remarkable performance and was evaluated using network classification or intrusion detection datasets. Many have also tested their architectures on real-world communication networks. In what follows, I discuss research work in six different sub-fields that relate to topics presented in this thesis:

- Network Intrusion Detections (NIDs) and current state-of-the-art.

- Challenges and solutions for NIDs with encrypted network traffic.
- Analysis and feature spaces for NIDs.
- ML model optimization for NIDs.
- Generative models and data balancing for IDS training.
- Network flow counting and identification

## 4.1 Attack Detection

IDSs are a critical component of any network's cyber security arsenal. Their primary task, as the name implies, is to assist in detecting unauthorized network intrusions [40]. In general, there are two types of intrusion detection systems: 1) signature-based IDSs and 2) anomaly-based IDSs [135]. Signature-based IDSs rely on pattern matching to detect (known) attacks. There are a number of commercial and open-source signature-based IDSs available, such as SNORT [205]. Although signature-based IDSs can provide strong protection against known threats, they have their limitations, most notably their inability to detect unknown attacks or known attacks with evolving patterns. On the other hand, anomaly-based IDSs are an appealing alternative because they can overcome the aforementioned limitations. As discussed, such methods employ ML techniques to learn a detection model from available network datasets and then use it afterward to detect outliers. As many of those do not rely on signatures (e.g., UL, RL), they can detect zero-day attacks [135]. They do, however, rely on the availability of large and diverse datasets to learn the detection model.

## 4.2 Analysis of Encrypted Network Traffic

Recently, some works delved into the implications of encryption from analytical perspectives. In [234], the authors provide a survey of methods for encrypted traffic classification and analysis, but mostly from an application layer level. The authors present an overview of deep learning for encrypted traffic analysis in [200], covering works published in the last three years. Reviewed works successfully apply GANs, AEs, CNNs and LSTMs for protocol identification and traffic classification, but only one is reported for intrusion detection [243]. However, the authors evaluate their methods with the old Darpa'98 and ISCX2012 datasets, which are not encrypted (or, if existent, it is within the application layer). In general, last years' research on deep learning and advanced ML for encrypted traffic consider encryption at application layers and mostly for identifying protocols and applications (e.g., [156]), also obviating the discussion about features or assuming the capability of neural networks to extract features. In this regard, it is worth emphasizing the authors' reasoning in [234] when they state: "we also discovered that the use of information from the unencrypted parts of encrypted connections for a network anomaly detection is only briefly investigated by researchers". Here they indirectly point to the study of available features (statistics, headers).

## 4.3 Feature Spaces

### 4.3.1 The Importance of Feature Engineering

Key features are essential for solving any analytical problem. Without proper features, you cannot conveniently dissect input spaces in a way that the aimed classes are clearly isolated. The authors in [71] express this idea with regard to ML. They state that “determining which features to use is the most important factor of a successful ML algorithm". As for the intersection between ML and NTA, the authors in [153] emphasize this point as the first of three main challenges, which are: 1) key feature selection, 2) finding the best analysis algorithms, and 3) obtaining representative datasets for training and testing algorithms. In the same work, the authors insist on the relevance of investigating the discriminative power of features. However, the variety of features that can be used for flow analysis is immense. A quick look at the IP Flow Information Export (IPFIX) flow information elements defined by Internet Assigned Numbers Authority (IANA) shows an overwhelming amount of features [57], and this only shows a fraction of the many possibilities. Also, no formal knowledge links known attacks with flow/contextual/behavioral features or profiles. However, for well-known, obvious attack classes (e.g., scanning, Denial of Service (DOS)), and perhaps for the case of *packet length*, there have been many papers recognizing it as a key feature for the discrimination of such attacks in different works [79, 153, 172, 186].

### 4.3.2 Interdependence Between Algorithms and Feature Spaces

Depending on the type of classifier used, the representation of network traffic might also be affected. The authors in [103] use RNNs in an effort to preserve the temporal essence of network traffic. The Authors use an LSTM-based classifier, in which only a single packet is fed into the classifier at each iteration; in particular, a sequence of packet header features. Their architecture shows outstanding performance with the benefit of a classification on-the-fly, which means a decision is made after each packet. CNNs, known for their breakthrough performance in computer vision applications, have also been adopted and used to classify internet traffic. Nonetheless, presenting the traffic to the CNN is still challenging because these networks are designed for unstructured data (i.e., images) whereas network traffic is mostly tackled in structured data formats, thus needs special transformations. The authors in [152] convert packet payloads into images aiming to discover patterns in the application layer using CNNs. The authors in [244] and [248] use CNNs to extract features which is known in the literature as representation learning [28]. These vector representations are subsequently used as a new dataset that permits traditional traffic classification methodologies. Similarly, in a hybrid combinations with the RNN, LSTM and Gated Recurrent Network (GRU) networks, the authors in [237] use and apply CNNs to individual application layer features of each packet in a flow. An AE was used in conjunction with PCA for input space transformation by [249]. The flow-based representation was replaced by a sub-space representation and fed into CNNs for classification. The authors in [54] use CNNs as a feature extractor to excerpt representations built by the first few convolutional layers and afterwards used k-NN to combine sample clusters together and SVM to perform classification. Finally, in [150], the authors carry out a clever feature crafting technique by translating different packet features into binary vectors, combining them into eight-bit pixels (one byte), and then converting them into images prepared for a regular CNN classification. Sometimes however, feature engineering can be skipped altogether, Deep Neural Networks (DNNs) provided the option of skipping manual crafting of statistical features because the latter can internally construct a latent space when fed with the appropriate data. Works like [91, 228, 4, 187, 157] use DNNs for either classification or detection tasks with automatic feature map construction and achieve notable performance. Nonetheless, network traffic can not always be represented with mere numerical data and still involves minimal space transformation.

Previous studies have exhibited considerable diversity and minimal consensus on how to represent network traffic adequately. Although recent trends have emphasized the use of unstructured data and the application of deep learning techniques to automatically extract features, there is no evidence (in the context of NTA) that these initiatives have outperformed conventional methods of representing traffic i.e., as feature vectors constructed with statistical values derived from packets and flows.

### 4.3.3 Feature Vectors Commonly Used

Several studies focus on the impact of features on attack detection and traffic classification. For example, the 41 base features present in the most popular and frequently used IDS datasets for the last two decades (Darpa-KDD'98, Darpa-KDD'99, and NSL-KDD [121]) are studied in [117]. Their discriminant power and extraction costs are discussed, concluding with a reduced subset of 16 outstanding features. This set of features might be deemed incidental nowadays because network traffic technologies, protocols, uses, and threats have evolved and changed since these datasets were collected and generated. In this regard, the authors in [175] provide a complete list of 248 features to be used for traffic classification and attack detection. Such a list collects all the features traditionally used with the mentioned datasets and in the most cited research in the field. Further, the authors in [85] perform a meta-study, which provides several features repeatedly used in recent research papers. This set is subsequently mentioned in [172] and is branded as the *Consensus* feature vector. Many similar Statistical feature vectors are now commonly used for testing new algorithms, e.g., [246, 118, 119]. The authors of [14] move a step further and construct a fine-grained vector with around 650 features which uses application layer features (TLS, HTTP, Domain Name System (DNS)) and obtain very good classification performance for detecting malware in TLS-encrypted traffic. Unfortunately, many of the used features are not available in most IDS datasets, which either complicates its use in comparative studies or significantly downgrades its performance [115].

### 4.3.4 Feature Vectors Used Throughout This Thesis

In line with the observations above, we tackle the optimization of feature spaces for attack detection in [172], including discussions about the implications of encryption. In the same work, we study and compare five representations using the UNSW-NB15 dataset provided by [178]. we conclude that host-based profiling has the best performance [119], and a derived vector obtained from a meta-analysis by the authors in [85] gives the best complexity-accuracy trade-off.

In the course of this work, I often refer to and use four feature vectors obtained from the literature, namely:

- The vector proposed by [246], referred to as CAIA, has been frequently used (with minimal variations) in traffic classification and attack detection with ML.
- The TA vector by [118], used to identify main patterns in backbone Internet traffic according to unidirectional, straightforward time series footprints.
- The AGM vector by [119], designed for discovering profiles in the Internet Background Radiation and consisting on observing flows that are defined to capture the behavior of network devices.
- The Consensus vector by [85], uses a meta-analytical approach and establishes the recommended vector after weighting the features used in the most prominent research conducted from 2002 until 2017.

As an example, Table 4.1 shows the CAIA feature vector [246]. It consists of 12 features, 7 of which are measured in both directions and 2 of which are expanded using four statistical combinations (i.e. $A$ becomes $mean(A)$, $min(A)$, $max(A)$ and $stdev(A)$).

**Table 4.1:** CAIA flow representation.

| Direction | Features | Statistical Operations |
|---|---|---|
| No direction | flowDurationMilliseconds<br>sourceTransportPort<br>destinationTransportPort<br>protocolIdentifier<br>octetTotalCount | None |
| Forward and Backward | ipTotalLength<br>interPacketTimeSeconds | Mean, Min, Max, Stdev |
| | packetTotalCount<br>tcpSynTotalCount<br>tcpAckTotalCount<br>tcpFinTotalCount<br>tcpCwrTotalCount | None |

## 4.4 Neural Network Optimizations for Faster Attack Detection

In the context of optimizing neural networks for attack detection, I refer to the concept of *cascading* classifiers, which means encompassing several classifiers that are used in a cascade (like a chain) as proposed by [239, 252, 242]. If the accuracy of a classifier in the beginning, is good enough, then there is no need to evaluate the other classifiers in the chain, which saves computational resources. This approach was predominantly explored for classical ML methods. The two possibilities are to place all classifiers in a row or the shape of a tree. Different classifiers can be used at each step if a tree is used, depending on which features are probably relevant for the sample currently under investigation. The authors in [34] aim to reduce the computational complexity during evaluation for large CNNs using an additional classifier after each layer. These additional classifiers learn to look at the previous layer's output and decide whether it is necessary to continue evaluating the next layers. The author mentions the approach most similar to what is discussed in Part III - Chapter 3 in [148, 147]. The authors introduce an additional output layer after each stage and then decide whether the next layers should be evaluated based on whether the confidence of the current output layer is high enough. They also argue that if many layers are required, the computation can be offloaded to a more powerful machine in the cloud, which saves further computational effort. Another line of work in [215, 257, 98, 23] focuses on training RNNs to skip irrelevant parts of the input sequence.

## 4.5 Generative Models for Network Data Balancing

GANs were originally developed for synthetic image generation [97]. The quality of images generated with original architectures was low, and they were easily distinguishable from real images as investigated by the authors in [191]. In recent years, however, GAN-architectures, such as StyleGAN, have become more sophisticated, making it increasingly hard to tell if generated samples are real or fake [129]. Since their inception, GANs have been applied successfully to numerous other tasks, such as generating video, audio, text, tabular data, and even entire video games [231, 72, 77, 181, 251, 137]. The success of GANs has also attracted the attention of the cyber security research community. GANs were recently also used to generate network traffic on a flow level [203] or on a packet level [53] and to generate both normal traffic [218] as well as synthetic malicious traffic [50, 154].

On the other hand, network traffic generation has been a long-standing issue due to a shortfall of labeled traffic for research purposes. Many works used techniques other than GANs to solve this problem, including [240, 36, 247, 95, 143, 261], and even proposed methods to control the generation process [224]. Some proposed Deep Learning (DL)-based solutions that aid in balancing minority classes to deal with class imbalance; an issue frequently encountered in anomaly detection datasets [106, 258, 125].

## 4.6 Network Flow Counting

“Flow counting" was introduced as a problem to count non-encrypted flows in high-speed links, deal with the growing size of hashing tables, and perform fast look-ups while capturing additional traffic. The authors in [136] propose a solution that uses timestamps and their delta to estimate the number of flows by also leveraging Bitmap algorithms, which shows high performance and low error. Similarly, the authors in [213] propose an alternative solution based again on Bitmaps [78] and use a sliding window to count flows in non-encrypted traffic over high-speed links. The authors in [110] use sampling and estimation to count the number of flows as a viable approach if the per-packet inspection is not feasible or requires specialized hardware. However, tunneling makes the previous approaches not suitable. This is because flows are aggregated together when leaving a local network or even a host, making counting the number of flows using flow keys, for example, no longer applicable since the same key can encapsulate a variable number of flows. Nevertheless, *knowing the number of flows* is a prerequisite assumed in many applications related to network traffic analysis. For instance, a solution to identify Youtube flows in encrypted traffic in proposed in [149]. The number of flows is needed to differentiate between video and audio flows. This is particularly determining when Quick UDP Internet Connection (QUIC) is used, and both traffic is sent over the same end-to-end connection. The authors in [25] present an encrypted traffic classification scheme that classifies network applications using clustering. The flow count is an optimal estimate of the number of centroids because, as the authors defined it, it is a multiple of the number of protocols and/or applications. Otherwise, if the number of flows is unknown, the number of clusters must be roughly guessed by means of iterative internal validation techniques, which are hardly feasible in actual implementations. In [217], the authors present a method for identifying “Anonymity" flows in encrypted traffic at the packet-, flow-, and user-levels. In their experiments, authors collect flows aggregated by the host. Thus, for users that handle multiple network services simultaneously on a single host, the flow count is required to identify “Anonymity" flows. The authors in [107] propose a method for identifying mobile applications in encrypted traffic. They show that different applications generate traffic with different patterns but also remark that encryption causes problems when two applications are simultaneously used. If the number of flows is known, traffic can be cleaned by removing interfering applications, overcoming the mentioned limitation. Overall, many proposed network traffic analysis methods require the number of concurrent flows either as a parameter in the analysis framework or indirectly to clean captured traffic beforehand and remove interfering flows [52, 214, 74, 66].

In Part II - Chapter 3, I address the same problem and propose a new method for counting flows in encrypted network tunnels using minimal information and leveraging LSTMs.

# Part II

# Feature Spaces

# Overview

One of the pillars for successful anomaly detection using ML is the proper extraction of knowledge from network data. Several researchers proposed representations and feature vectors to accomplish this task over time (refer to Part I - Chapter 4). Some of these are compared in [172]. In this first primary part, I present feature vectors for representing network traffic in a shape that is easily handled by most ML algorithms. To be more specific, this part consists of:

**Investigation of a novel feature space for flow-based detection in unencrypted and encrypted traffic.** Here, I present the *multi-key* feature vector, a network traffic representation based on a new aggregation approach for network packets. The proposal is evaluated with a RF classifier for the task of attack detection. The implications of encryption as well as explainability aspects are studied as well through extensive experimentation.

**Investigation of feature spaces for packet-sequence-based detection.** In scenarios where a prompt reaction is required, flow aggregation and processing might be too slow and costly. I propose I-Notice to overcome this limitation. In comparison to the *multi-key* representation, I show that, while the latter performs very well, it is not the only viable option, particularly in certain circumstances (e.g., when a prediction is needed at a packet level). I-Notice uses CNNs for detection by processing packet header values as input channels and incorporate incremental learning for increasingly adding classes into the learned model without retraining from scratch. Continuous learning is required in most network IDSs nowadays to keep up with the changing nature and the increasing number of network attacks. Finally, I examine the framework's explainability and discuss potential adversarial attacks.

**Investigation of detection possibilities for aggregated encrypted flows.** I explore other network traffic representations by using only a few features to solve a task involving counting the number of flows in VPN tunnels. A proposed representation is evaluated and tested in conjunction with a LSTM network. By constructing a suitable input space, I show that it is possible to infer the number of aggregated flows. Later, I go over the explainability aspect and potential adversarial attacks.

Chapters 1, 2 and 3 primarily discuss new representations to solve various sub-tasks related to network anomaly detection. In Chapter 4, I provide an **investigation of transformation and scaling methods applied to network traffic** for the sole purpose of anomaly detection. In the used datasets, I notice that the data follows a specific distribution that may not be compatible with a variety of candidate algorithms used subsequently. As a result, I perform experiments to assess the impact of data transformation, scaling, and normalization on data quality and the learned model. These manipulations can change the data distribution, shape, and scale; and often break colinearities between features, so they must be done with caution. All of these points are covered in Chapter 4.

Finally, I briefly summarize the set of datasets and algorithms used in each chapter in Table 0.1.

| Chapter | Datasets | Algorithms |
|---|---|---|
| 1 | CICIDS2017, UNSW-NB15, ISCX-Bot-2014 | RF, SHAP |
| 2 | CICIDS2017, UNSW-NB15, MAWI | CNN, SHAP, iCarl, MLP |
| 3 | MAWI | LSTM, PDP |
| 4 | CICIDS2017, UNSW-NB15, ISCX-Bot-2014, MAWI | MLP, NB, DT, kNN |

**Table 0.1:** A summary of datasets and algorithms used in each chapter.

CHAPTER 1

# Flow-based Feature Vectors with the Multi-key Approach

***Notice of adoption from previous publications - 3***
*Parts of this chapter's content have been published in the following paper:*
*Fares Meghdouri, Félix Iglesias Vázquez, and Tanja Zseby. Cross-layer profiling of encrypted network data for anomaly detection. In* 2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 469–478. IEEE, 2020*

As discussed previously, the selection of features for attack detection and network traffic classification is still an open question in the scientific community. This problem is further exacerbated by the fact that the implications of encryption in feature selection processes or in the presentation of novel detection vectors are commonly forgotten or obviated.

## 1.1 Introduction

In this chapter we propose a novel flow-feature vector that combines different traffic representations that have shown satisfactory performance results in previous works. Our proposal is named *multi-key* as it condenses three alternative ways of modeling network traffic: *application*, *conversation*, and *endpoint* profiles. We additionally study the compatibility with the two most utilized protocols for encrypting modern Internet traffic at network and transport layers: TLS and IPsec. We leave apart QUIC since it is still in a development phase, specifications change continuously, and the technology is not consolidated yet. For the evaluation of this approach, we run experiments over three IDS datasets and use detection algorithms based on RF. DT-based learners are the supervised classifiers that tend to show the most accurate, robust, and explainable performances in the field, e.g., [189, 234, 178]. In addition, we use RF since they provide a better and simpler explanation of predictions compared to other supervised learning techniques based on DL and to compare performance with other studies.

Experiments described in Section 1.3 show that when the *multi-key* vector is applied, attack detection is outstanding regardless of TLS encryption. The higher coverage of IPsec causes performance drops for any detector, but the *multi-key*-based approach still reaches satisfactory results when identifying botnets hidden under IPsec tunnels. Results support the triple perspective of the *multi-key* vector as a step forward in the design of new data representations with higher discriminant power, and therefore an excellent baseline for future NIDSs. Summarizing, in this part we:

- **propose** the *multi-key* vector, a novel cross-layer representation of network traffic data that profiles traffic from *application*, *conversation*, and *endpoint behavior* perspectives.

- **test** the *multi-key* vector for attack detection with three IDS datasets published in 2014, 2015, and 2017, and compare its performance with previous works.

- **study** the impact of TLS and IPsec encryption in network attack detection when using the *multi-key* vector to represent traffic flows.

## 1.2 Proposal: Multi-Key Feature Vector

Previous works suggest that the combination of different feature vectors might lead to enhanced attack detection performances [172]. We further explore this idea by bringing together diverse flow-based representations from existing literature. We obviate the ones that are obsolete, use obscure or unpublished extraction methods/models, use features that are too costly or not clearly defined, demand deep packet inspection, or require information not available in the data due to network restrictions. Therefore, we reduce the scope and build a new mixed representation based on CAIA, TA, AGM and Consensus feature vectors (introduced in Section 4.3). Constructing the cross-layer vectors has implications that we discuss as follows.

### 1.2.1 Combining traffic representations

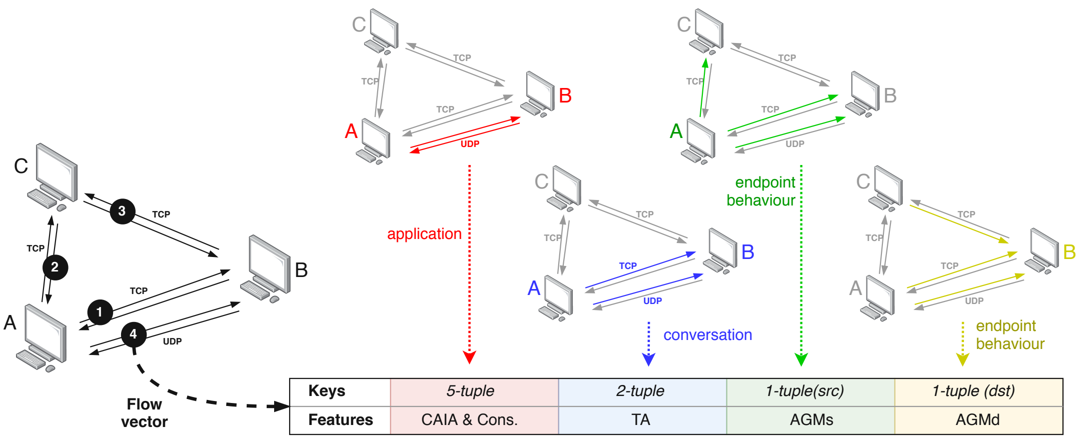
Merging different traffic vectors is not a straightforward task. The CAIA, TA, AGM, and Consensus formats use different flow-keys (see Table 1.1), some being uni-directional and others bi-directional. To make such representations compatibles, we join information extracted from each variant by intersecting flow-keys in a rigorous way.

| **Key Feature** | **CAIA & Consensus** | **TA** | **AGM$_s$** | **AGM$_d$** |
|---|---|---|---|---|
| sourceIPAddress | ● | ● | ● | . |
| destinationIPAddress | ● | ● | . | ● |
| sourceTransportPort | ● | . | . | . |
| destinationTransportPort | ● | . | . | . |
| protocolIdentifier | ● | . | . | . |

**Table 1.1:** Flow keys of Consensus, CAIA, TA, AGM$_s$ (profiling **s**ources), and AGM$_d$ (profiling **d**estinations) vectors.

Note that, when flows are defined/aggregated based on the traditional 5-tuple (sourceIPAddress, destinationIPAddress, sourceTransportPort, destinationTransportPort, protocolIdentifier), data samples capture information about the *application*. Instead, if the 2-tuple (sourceIPAddress, destinationIPAddress) is used, data samples profile *conversations* between two hosts. Finally, the 1-tuple (either sourceIPAddress or destinationIPAddress) implies data samples collecting *endpoint behaviors*. In the *multi-key* feature vector, the 5-tuple is kept as a basis, and the information extracted using the 2-tuple and 1-tuple keys are subsequently added as a sub-aggregation.

Table 1.2 shows the features that form the *multi-key* vector. The AGM vector is used twice: taken sourceIPAddress as flow key (AGM$_s$) and taken destinationIPAddress as flow key (AGM$_d$). This allows profiling endpoints as information senders and receivers respectively. Profiling endpoints is very useful to identify some attack-related behavioral schemes; for instance, scanners or victims sending back-scatter (senders), or victims under Denial of Service (DoS) (receivers).



| Keys | 5-tuple | 2-tuple | 1-tuple(src) | 1-tuple (dst) |
|---|---|---|---|---|
| Features | CAIA & Cons. | TA | AGMs | AGMd |

**Figure 1.1:** Example of the *multi-key* construction. In the example network, hosts A, B, and C exchange packets through different applications. Given a flow vector (e.g., vector number 4), it contains information about the application (in red), the conversation (in blue), and the behaviors of the source and destination (in green and yellow respectively).

The example in Figure 1.1 can help understand how the *multi-key* vector is created. In the example, we can see a small ad-hoc network with three hosts —A, B, and C— that communicate with each other. For a predefined observation time, communications are captured in four different flows. Figure 1.1 focuses on flow number 4, which corresponds to the UDP-based communication between A and B, started by A and with predefined port numbers. The traditional 5-tuple key is used as a “preliminary" key to extract CAIA and Consensus features. Note that the same vector additionally is appended with information related to the whole *conversation* between A and B (by using a 2-tuple key and TA features) and therefore ignores the concept of port number. Finally, information about the *endpoint behaviors* of A as a sender and B as a receiver is also stored in the same flow vector which consists of the entire traffic going from or into A and B.

The power of the *multi-key* approach lies in collecting information related to the application, the conversation, and the behavior of endpoints at the same time and in the same sample; in other words, they store: single application statistics (CAIA and Consensus), conversation statistics (TA) and single host (attacker or victim) statistics (AGM). Detectors obtain a deeper insight of the analyzed traffic and can profile threats based on the nature of the specific application, the step-by-step strategy described in the conversation, the global behavior of an attacker or a victim in the network, or all these aspects at the same time. This allows to detect patterns that can not be otherwise detected by using each representation separately.

### 1.2.2 Reducing Redundancy

Since network data features are known to be highly correlated [117], merging different traffic representations is prone to involve a strong redundancy (multicollinearity or high feature correlation). Hence, after joining the different formats, feature selection is required to keep only relevant features. In this respect, RFs have shown to be robust feature selection filters and able to evaluate feature importance during classification [208].

### 1.2.3 Dealing With Encryption

As mentioned is Section 3.4, encryption prevents extracting features from certain layers. For instance, with TLS, all features encapsulated above the transport layer are not accessible. Respectively, with IPsec, features encapsulated above the network layer are not available either. This has some implications for the *multi-key* feature vector:

- No constraints for the TLS case, i.e., fully compatible.
- The 5-tuple flow-key cannot be built for the IPsec case. Additionally, all features extracted from the transport layer are unavailable. A reduced version of the *multi-key* vector only using the 2-tuple key is therefore used in this case.

The right columns of Table 1.2 indicate which features are available for the complete vector (not encrypted or TLS), and for the reduced variant (with IPsec).

**Table 1.2:** Multi-key vector features. The additional three columns reflect whether a feature is available with TLS encryption, IPsec encryption (transport mode) or removed before the analysis either because it shows a sparse distribution or it is irrelevant for the classification task. The feature naming and syntax is based on the IPFIX standard [58]. A few features/functions are newly introduced: (a) "apply" means applying the first function into the second attribute, (b) "modeCount": it is the number of packets sent to the most recurring source or destination host, (3) "NTA( X )": features introduced in [118] and "forward/backward" refers to the set of packets in one of the directions based on the first observed packet.

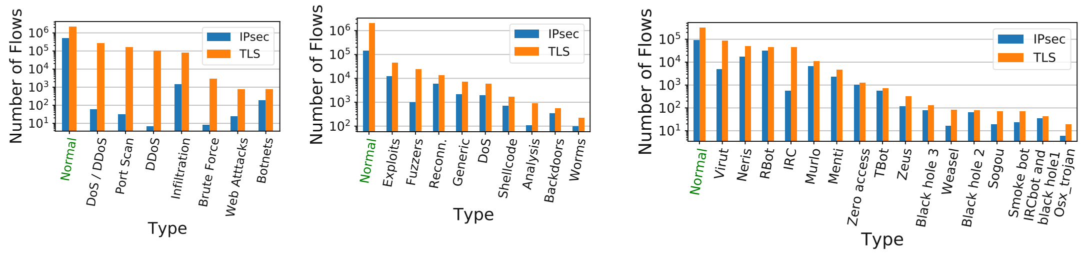
| | **Feature** | **TLS** | **IPsec** | **Removed** |
|---|---|---|---|---|
| General features | flowStartMilliseconds | ● | ● | ● |
| | flowDurationMilliseconds | ● | ● | . |
| | sourceIPAddress | ● | ● | ● |
| | destinationIPAddress | ● | ● | ● |
| | sourceTransportPort | ● | . | ● |
| | destinationTransportPort | ● | . | ● |
| | protocolIdentifier | ● | . | . |
| From CAIA & Consensus | apply(packetTotalCount,forward) | ● | ● | . |
| | apply(octetTotalCount,forward) | ● | ● | . |
| | apply(min(ipTotalLength),forward) | ● | ● | . |
| | apply(max(ipTotalLength),forward) | ● | ● | . |
| | apply(median(ipTotalLength),forward) | ● | ● | . |
| | apply(mean(ipTotalLength),forward) | ● | ● | . |
| | apply(mode(ipTotalLength),forward) | ● | ● | . |
| | apply(stdev(ipTotalLength),forward) | ● | ● | . |
| | apply(min(interPacketTimeSeconds),forward) | ● | ● | . |
| | apply(max(interPacketTimeSeconds),forward) | ● | ● | . |
| | apply(median(interPacketTimeSeconds),forward) | ● | ● | . |
| | apply(mean(interPacketTimeSeconds),forward) | ● | ● | . |
| | apply(variance(interPacketTimeSeconds),forward) | ● | ● | . |
| | apply(tcpSynTotalCount,forward) | ● | . | . |
| | apply(tcpAckTotalCount,forward) | ● | . | . |
| | apply(tcpFinTotalCount,forward) | ● | . | . |
| | apply(tcpCwrTotalCount,forward) | ● | . | . |
| | apply(packetTotalCount,backward) | ● | ● | . |
| | apply(octetTotalCount,backward) | ● | ● | . |
| | apply(min(ipTotalLength),backward) | ● | ● | . |

Continued on next column

**Continued from previous column**

| | Feature | TLS | IPsec | Removed |
|---|---|---|---|---|
| | apply(max(ipTotalLength),backward) | ● | ● | . |
| | apply(median(ipTotalLength),backward) | ● | ● | . |
| | apply(mean(ipTotalLength),backward) | ● | ● | . |
| | apply(mode(ipTotalLength),backward) | ● | ● | . |
| | apply(stdev(ipTotalLength),backward) | ● | ● | . |
| | apply(min(interPacketTimeSeconds),backward) | ● | ● | . |
| | apply(max(interPacketTimeSeconds),backward) | ● | ● | . |
| | apply(median(interPacketTimeSeconds),backward) | ● | ● | . |
| | apply(mean(interPacketTimeSeconds),backward) | ● | ● | . |
| | apply(variance(interPacketTimeSeconds),backward) | ● | ● | . |
| | apply(tcpSynTotalCount,backward) | ● | . | . |
| | apply(tcpAckTotalCount,backward) | ● | . | . |
| | apply(tcpFinTotalCount,backward) | ● | . | . |
| | apply(tcpCwrTotalCount,backward) | ● | . | . |
| From AGM(source/destination) | distinct(destinationIPAddress/sourceIPAddress) | ● | ● | . |
| | mode(destinationIPAddress/sourceIPAddress) | ● | ● | ● |
| | modeCount(destinationIPAddress/sourceIPAddress) | ● | ● | . |
| | distinct(sourceTransportPort) | ● | . | . |
| | mode(sourceTransportPort) | ● | . | ● |
| | modeCount(sourceTransportPort) | ● | . | . |
| | distinct(destinationTransportPort) | ● | . | . |
| | mode(destinationTransportPort) | ● | . | ● |
| | modeCount(destinationTransportPort) | ● | . | . |
| | distinct(protocolIdentifier) | ● | . | . |
| | mode(protocolIdentifier) | ● | . | . |
| | modeCount(protocolIdentifier) | ● | . | . |
| | distinct(ipTTL) | ● | ● | . |
| | mode(ipTTL) | ● | ● | . |
| | modeCount(ipTTL) | ● | ● | . |
| | distinct(tcpFlags) | ● | . | . |
| | mode(tcpFlags) | ● | . | . |
| | modeCount(tcpFlags) | ● | . | . |
| | distinct(octetTotalCount) | ● | ● | . |
| | mode(octetTotalCount) | ● | ● | . |
| | modeCount(octetTotalCount) | ● | ● | . |
| | packetTotalCount | ● | ● | . |
| | NTAFlowID | ● | . | ● |

Continued on next column



**(a)** CICIDS2017 dataset **(b)** UNSW-NB15 dataset **(c)** ISCX-Bot-2014 dataset

**Figure 1.2:** Class distributions in logarithmic scale when extracting flow using both variants of the *multi-key* feature vector (TLS and IPsec).

**Continued from previous column**

| | **Feature** | **TLS** | **IPsec** | **Removed** |
|---|---|---|---|---|
| From TA | NTAProtocol | ● | . | . |
| | NTAPorts | ● | . | ● |
| | NTATData | ● | ● | . |
| | packetTotalCount | ● | ● | . |
| | count(NTASecWindow) | ● | ● | . |
| | divide(NTATData,count(NTASecWindow)) | ● | ● | . |
| | divide(packetTotalCount,count(NTASecWindow)) | ● | ● | . |
| | max(NTATOn(NTASecWindow)) | ● | ● | . |
| | min(NTATOn(NTASecWindow)) | ● | ● | . |
| | max(NTATOff(NTASecWindow)) | ● | ● | . |
| | min(NTATOff(NTASecWindow)) | ● | ● | . |
| | count(NTATOn(NTASecWindow)) | ● | ● | . |

## 1.3 Evaluation

In this section, we describe the conducted experiments to evaluate our proposal. We introduce the used datasets, the classification test-bed, and the evaluation metrics. Scripts are available for further exploration and replication either online[1] or by contacting the author.

### 1.3.1 IDS Datasets

To evaluate the performance, we use three IDS datasets: CICIDS2017, UNSW-NB15 and ISCX-Bot-2014. Figure 1.2 uses different colors (orange and blue) to represent the number of extracted flows when using each of the *multi-key* variations: complete (with TLS) and reduced (with IPsec). Obviously the IPsec variant contains fewer flows due to a more coarse-grained aggregation.

[1]`github.com/CN-TU/multi-key-vector-experiments`

### 1.3.2 Classification Test-bed

The experimental test-bed is similar to the generic scheme introduced in Part I - Section 2.1, it: (a) retrieves raw pcaps as inputs, (b) extracts flows in vectors according to pre-configured features, (c) joins multiple flow vectors to form the *multi-key* feature set, (d) labels flow-instances based on ground truth files (for evaluation purposes), (e) performs transformations, data splits, cross-validation, parameter tuning, feature selection, and data reduction, (f) analyzes and classifies preprocessed flows with supervised learners, and (g) evaluates classification results and provides performance results.

The experimental test-bed can be split in three main blocks:

1. Extraction.

   In this phase we used go-flows [241] to extract information from raw traffic captures (pcaps) into network flows. Flows are built and joined together to form the *multi-key* vector introduced in Section 1.2. Finally, instances are labeled based on the ground truth files (provided by the data sources) to allow subsequent evaluation. To allow for further analysis, two kinds of labels are used: “attack name” (categorical) and “malicious/non-malicious” (binary).

2. Preprocessing.

   - *Train-test separation.* Data instances are split into a training set (70%) and a test set (30%). We applied stratified sampling to keep equivalent class proportions.
   - *Standardization.* Data is z-score normalized. This is required to optimize posterior PCA transformations.
   - *Feature selection.* Irrelevant features (i.e., features not used during the classification) are removed with RF.
   - *Feature reduction.* We apply PCA to shrink the resulting space (whenever possible) with minimal to no information loss. This step makes further analysis with RFs faster and simpler [111].

3. Analysis.

   - *Parameter tuning* of the RF classifier with a genetic search based on maximizing the *F1* score.
   - *Cross-validation.* Even in spite of the fact that RF naturally minimize overfitting with suitable parameters, a 5-fold cross-validation is performed to reinforce generalization.
   - *Evaluation.* Performance indices are obtained based on the metrics defined in Section 1.3.3.

### 1.3.3 Evaluation metrics

We evaluate performances by using Roc-Auc and F1 scores for consistency with previous works. Some works have emphasized that the Prec-Rec (Precision-Recall) curve is better suited than the Roc curve when evaluating binary classification with unbalanced classes, e.g., [67, 209]. Therefore, we additionally provide the Average Precision (A-Precision), which can be computed from the Prec-Rec curve [38]. We refer readers to [38] and [82] for further information about the Prec-Rec curve and the Average Precision metric.

### 1.3.4 Explainability

In order to better understand the power of the *multi-key* feature as a profiling tool, we additionally analyzed features prior to space transformations by means of SHAP [160, 159]. To do this, we used the proposed tree explainer that computes *Shapley Values*, which represent the marginal contribution of each input feature to the final label. The tree explainer shows the average amount of change in the output when perturbing inputs. Such values allow to understand decisions made by the classifier and enable detection of overfitting and data artifacts.

## 1.4 Results and Discussion

In this part we discuss experimental results for TLS and IPsec encrypted flows and compare them with previous state of the art. As aforementioned, all experiments are publicly available for further tests and replicability[1].

### 1.4.1 Detection Performances

Table 1.3 collects performance scores when evaluating over three IDS datasets under test, with both encryption schemes TLS and IPsec. We show the mean and standard deviation across 5 cross-validation folds. Some insights can be inferred:

- As a general rule, the *multi-key vector obtains high detection rates*. Moreover, the selection of RF, the use of cross-validation, and the fact that training and test results are consistent guarantee robustness and reduce possible overfitting. Cross-validation shows low variance, suggesting stability in the learning models regardless of the datasets. Based on subsequent investigations on why some variants achieve a score of 100%, we found that the datasets are unfortunately not heterogeneous enough for the trained model to yield FPs and FNs. This, however, does not interfere with the results suggesting that our proposal is powerful.

- *The IPsec encryption considerably affects attack detection in general, but not botnet detection in the ISCX-Bot-2014 dataset.* By comparing TLS and IPsec variants, performance drop occurs for the IPsec case in the CICIDS2017 and ISCX-Bot-2014 datasets. This was expected because such encryption removes information that is

**Table 1.3:** Performance indices of conducted experiments with a RF classifier. Variants with 100% scores have 0 variance because samples in all folds were correctly classified, this is due to samples with trivial patterns being easily classified.

| **CICIDS2017** | **MD** | **MSL** | **Accuracy** | **Precision** | **Recall** | **F1-score** | **A-Precision** | **Roc-Auc** |
|---|---|---|---|---|---|---|---|---|
| *TLS – training* | 30 | 1 | 1.000 ±0.000 | 1.000 ±0.000 | 0.999 ±0.000 | 0.999 ±0.000 | 1.000 ±0.000 | 1.000 ±0.000 |
| *TLS – testing* | | | 1.000 ±0.000 | 1.000 ±0.000 | 0.999 ±0.000 | 0.999 ±0.000 | 1.000 ±0.000 | 1.000 ±0.000 |
| *IPsec – training* | 30 | 1 | 0.999 ±0.000 | 0.997 ±0.008 | 0.792 ±0.074 | 0.882 ±0.048 | 0.927 ±0.013 | 0.984 ±0.005 |
| *IPsec – testing* | | | 0.999 ±0.000 | 0.992 ±0.021 | 0.758 ±0.046 | 0.859 ±0.037 | 0.882 ±0.034 | 0.963 ±0.017 |

| **UNSW-NB15** | **MD** | **MSL** | **Accuracy** | **Precision** | **Recall** | **F1-score** | **A-Precision** | **Roc-Auc** |
|---|---|---|---|---|---|---|---|---|
| *TLS – training* | 26 | 2 | 0.994 ±0.000 | 0.939 ±0.002 | 0.934 ±0.003 | 0.936 ±0.002 | 0.981 ±0.002 | 0.999 ±0.000 |
| *TLS – testing* | | | 0.994 ±0.000 | 0.929 ±0.007 | 0.931 ±0.009 | 0.930 ±0.003 | 0.973 ±0.002 | 0.999 ±0.000 |
| *IPsec – training* | 19 | 6 | 0.994 ±0.001 | 0.965 ±0.008 | 0.992 ±0.002 | 0.978 ±0.003 | 0.987 ±0.006 | 0.999 ±0.000 |
| *IPsec – testing* | | | 0.994 ±0.001 | 0.965 ±0.008 | 0.993 ±0.005 | 0.979 ±0.004 | 0.989 ±0.006 | 0.999 ±0.000 |

| **ISCX-Bot-2014** | **MD** | **MSL** | **Accuracy** | **Precision** | **Recall** | **F1-score** | **A-Precision** | **Roc-Auc** |
|---|---|---|---|---|---|---|---|---|
| *TLS – training* | 30 | 1 | 0.997 ±0.000 | 0.999 ±0.000 | 0.993 ±0.001 | 0.996 ±0.001 | 1.000 ±0.000 | 1.000 ±0.000 |
| *TLS – testing* | | | 0.994 ±0.001 | 0.998 ±0.000 | 0.989 ±0.001 | 0.993 ±0.001 | 1.000 ±0.000 | 1.000 ±0.000 |
| *IPsec – training* | 25 | 2 | 0.996 ±0.001 | 0.998 ±0.001 | 0.991 ±0.003 | 0.995 ±0.001 | 1.000 ±0.000 | 1.000 ±0.000 |
| *IPsec – testing* | | | 0.994 ±0.002 | 0.997 ±0.002 | 0.989 ±0.004 | 0.993 ±0.002 | 1.000 ±0.000 | 1.000 ±0.000 |

MD: `max-depth`, MSL: `min-samples-at-leaf`. The `number-of-estimators` is set to the half of number of features. Other parameters take default values according to the Python scikit-learn implementation (`V0.20.2`).

| Candidate | CICIDS2017 | UNSW-NB15 | ISCX-Bot-2014 |
|---|---|---|---|
| *Multi-key* | 1.000 | 0.926 | 0.992 |
| da Silva Neto et al. | 0.970 | - | - |
| Aksu et al. | 0.980 | - | - |
| Meghdouri et al. | - | 0.849 | - |
| Nguyen et al. | - | 0.760 | - |
| Yin et al. | - | - | 0.705 |
| Tariq et al. | - | - | 0.920 |

**Table 1.4:** Performance comparison among other papers.

potentially useful for the identification of attacks. The performance drop is more acute for the CICIDS2017 dataset, whereas it is minor for the ISCX-Bot-2014 case. This suggests that the detection of botnets in the later dataset is less affected by IPsec encryption than other types of attacks. A plausible explanation is that botnet flows are transported on communication level and not application level which suggests that each flow is isolated in an IPsec tunnel and, therefore, the behavior captured by the reduced variant of the *multi-key* vector is not disturbed.

- *The multi-key vector captured generation artifacts in the UNSW-NB15 dataset.* The IPsec case surprisingly shows better performances than the TLS variant for the UNSW-NB15 dataset. After carefully checking this counter-intuitive result, we realized that this is caused by the design of the UNSW-NB15 dataset. The dataset seems to expect analysis from *application* perspectives (i.e., aggregations based on a 5-tuple key). However, this becomes problematic in our case because the same malicious IP address in the dataset perform all types of attacks, such behavior being very easy to spot when using the *endpoint* approach incorporated in the *multi-key* representation (see Figure 1.1) which results in a single flow with the same characteristics representing a set of attacks. This design aspect can be considered as a *defect* in the UNSW-NB15 IDS dataset, but also shows the benefits of using multi-perspective approaches for analyzing traffic[2].

Figure 1.3 and Table 1.4 show a comparison among works published between 2017 and 2019 [220, 62, 172, 180, 255, 229]. In short, the *multi-key* feature vector obtains satisfactory performances and outperforms previous proposals evaluated on the same datasets or research that uses the same feature vectors individually [172].

### 1.4.2 Analysis of Features

Figure 1.4 shows the top 5-features used by the RF classifier to distinguish "attack-related" from "non-attack-related" flows. The figure shows six plots corresponding to

[2]In a later version of the dataset, authors fixed this issue and updated the dataset.

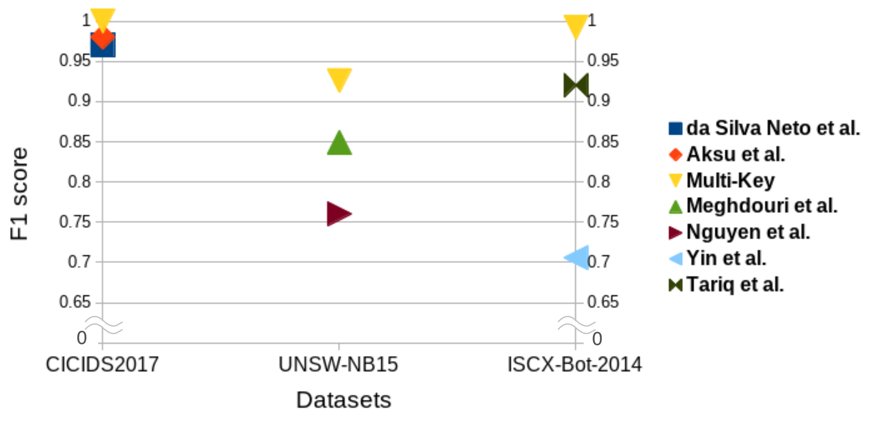


**Figure 1.3:** F1-score comparison of various proposals including da Silva Neto et al. [62], Aksu et al. [9], Meghdouri et al. [172], Nguyen et al. [180], Yin et al. [255], Tariq et al. [229].

the two studied encryption modes (TLS and IPsec) and the three tested IDS datasets (ISCX-Bot-2014, UNSW-NB15, and CICIDS2017).

Results show that there is a higher coincidence in the discriminant features discovered for the ISCX-Bot-2014 and CICIDS2017, whereas the UNSW-NB15 shows a more different feature set. This is consistent with the generation artifacts observed in the UNSW-NB15 dataset (discussed previously). Ideally, discovered feature sets should capture *necessary* patterns that are idiosyncratic of the used attack types, and not *contingent* mechanisms that belongs to the dataset generation setup. Therefore, coincidence in the top discriminant features through different IDS datasets is a good sign, though differences are expected because attack collections in each dataset are different, harmless or non-attack-related traffic is also differently generated, and, as emphasized before, correlation in network traffic features is usually strong.

As for the specific features, we observe that revealing detection patterns mainly combine:

- The volume of data sent by the client (in the conversation: "_NTATData" and in the application "octetTotalCount"), both in absolute values and in time-rates.
- The value and number of different TTL values ("ipTTL"). In general the TTLf field is useful to identify operating systems.
- The number of distinct destination IPs ("distinct (destinationIPAddress)") contacted by one host, which are key to identify scanners, machines performing active information gathering, or spreading malware.
- The most used TCP flag ("tcpFlags"), known to be essential components in specific attack schemes, e.g., TCP SYN Flood, Tsunami SYN Flood.

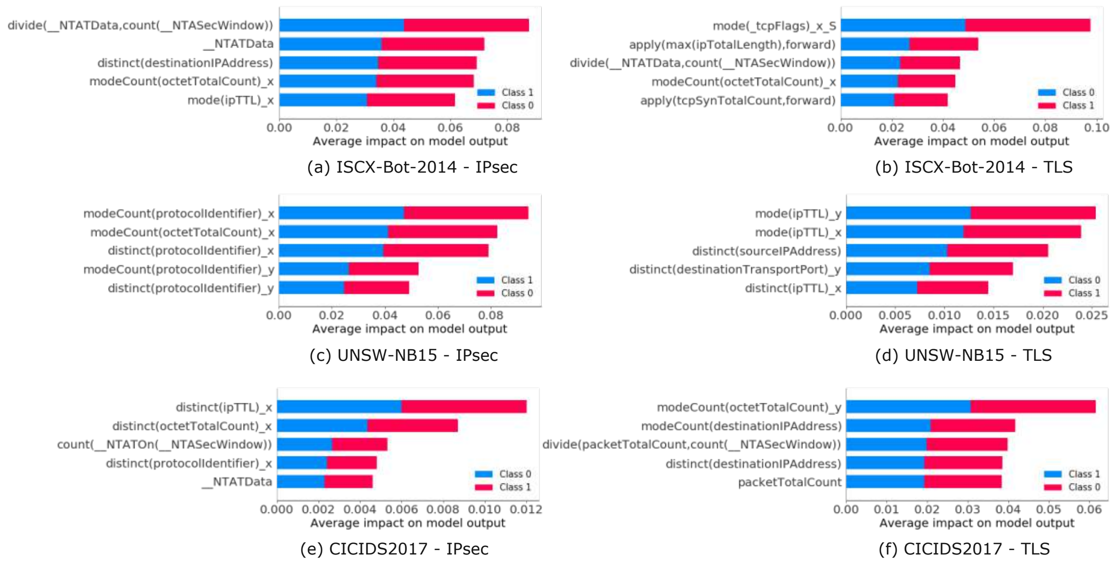


**Figure 1.4:** The five most relevant features sorted by the average impact on the prediction based on SHAP values. We analyze the contributions with each variant/dataset separately to allow better understanding of the decision mechanism of the RF classifier.

- The packet length (“ipTotalLength”), already considered as highly discriminant in previous works (see Part I - Chapter 4).
- The total number of packets (“packetTotalCount”). Highly correlated to the volume of sent data, it also characterizes the dynamics of the network traffic element regardless of such element being the application, the conversation, or the endpoint behaviour.
- The different used protocols (“protocolIdentifier”). Infiltration strategies commonly check out known vulnerabilities to enter the aimed machine, therefore involving various protocols. However, in the UNSW-NB15 case, it is likely disclosing a contingent pattern.

## 1.5 Conclusion

In this chapter, we have introduced the *multi-key* feature vector, a cross-layer approach to build network data representations for attack detection and traffic classification. The *multi-key* vector assembles state of the art feature vectors in a way that they are able to profile traffic at three different levels: application, conversation, and endpoint behavior. The *multi-key* vector draws more informative analysis spaces and leverages the fact that each type of network attack has a more suitable analysis perspective from which it can be better differentiated and detected.

We have tested our approach with three modern, last-generation IDS datasets (CI-CIDS2017, UNSW-NB15, and ISCX-Bot-2014), and studied the implications of IPsec and TLS encryption schemes for the analysis. In general, results show enhanced detection when compared to previous works. The *multi-key* vector can maintain high detection performances in spite of TLS encryption, but it considerably losses power with IPsec encrypted traffic. This excludes botnets attacks in ISCX-Bot-2014, which are satisfactorily identified even with IPsec encryption. We have studied the generated RF models using SHAP values and showed that the *multi-key* vector feature can adapt to different traffic classes, attack types and encryption schemes.

Modern attack detection in network communications currently faces serious challenges related to the analysis of big streams of encrypted data. The selection of network traffic features is an open question that deserves high attention due to the need for secure communications irrespective of the volume, speed, and data privacy mechanisms. We believe that the *multi-key* feature vector is a complete, lightweight, and flexible form to represent network traffic communications that allows the accurate profiling of traffic classes and attacks, therefore becoming the basis for effective ML-based network security and network data analysis.

CHAPTER 2

# Packet-based Feature Vectors with Mixed Packet Sequences

***Notice of adoption from previous publications - 4***
*Parts of this chapter's content have been published in the following paper:*
*Fares Meghdouri, Félix Iglesias Vázquez, and Tanja Zseby. Anomaly detection for mixed packet sequences. In* 2020 IEEE 45th LCN Symposium on Emerging Topics in Networking (LCN Symposium)*, pages 120–130. IEEE, 2020*

Given the evolving trend of pattern learning and data availability, numerous attempts published effective IDS architectures based on a broad range of ML techniques. As described in the previous chapter, solutions similar to the *multi-key* approach rely on the same foundation: (a) learn patterns, (b) predict a particular class or detect anomalies during deployment, and (c) take full or partial actions. To this end, network flows and session data are used as a baseline for representing data and focused on developing the core classifiers instead.

## 2.1 Introduction

Despite their exceptionally high efficiency and accuracy (see results obtained in Part II - Chapter 1 - Section 1.4), modern session-oriented representations and one-time-training detectors might face a couple of problems:

- Low-end hardware is known to be limited in computing power and memory capacity. Network devices like routers or switches cannot handle heavy preprocessing; despite their specialized capabilities of extracting standardized features such as IPFIX IEs. Such devices counter difficulties extracting custom features in an online manner especially in congested networks, namely because of the growing flow hash tables.

As an example, authors in [19] show that at a data-center switch, 3.8 PB of data is captured during only a 5 minutes period. Therefore, unless specialized hardware is installed, extracting complex features is a difficult task.

- Extracting data from packets yields more information than extracting data from aggregated packets i.e., network flows. Therefore it is of interest to use plain packets for detection and evaluate the performance of such approach.

- Pre-trained classifiers cannot be upgraded during deployment, so it is impossible to expand a model's capabilities i.e. add, update or delete existing classes only if a new classifier is established with both old and new training samples. This is somewhat a generic problem in ML. Nevertheless, if such a solution is opted for, a few problems arise, in particular the need for a longer training time if a classifier is trained from scratch each time. In addition, one should keep both old and new training samples which causes a rapid growth in storage requirements with the increasing number of classes.

In this chapter we propose a new architecture — I-Notice — to address both problems. We exploit One-Dimensional Convolutional Neural Networks (1D-CNNs) to obtain relevant information right off packet traces. Detecting abnormal traffic being our priority, we try to discover maliciousness in an input sequence without flow aggregation. In [144], authors show that CNNs are suitable for artifact detection in both unstructured and structured data whereby the position of the artifact is not relevant. Owing to their shared parameters, CNNs are able to achieve the former by applying a sliding convolution filter to the input sequence and therefore maximizing the convolution when a match is found. In our scenario, we define a "sequence" as the set of network packets acquired directly from a network interface and we consider the "artifacts" to be malicious packets with their specific properties (features) that belong to a specific malicious flow. This unique capability enables the classification of packet without distinguishing the different flows/connections the packet belong to and consequently reduce computing and buffering resources considerably.

With regard to the second problem, that is, lifelong learning, upon initial training and implementation we use an additional incremental training strategy to update the classifier continuously. The strategy allows the neural network weights and biases to be updated dynamically as the behavior of traffic changes over time and more classes are discovered, which we assume to be a pertinent characteristic of network traffic.

Ultimately, feeding mixed packages to the framework instantly opens up opportunities for faster online operation and decreases preprocessing cycles, which can lead to fewer deployment and aggregation failures.

Similarly to the *multi-key* proposal, we test the reliability of I-Notice with TLS and IPsec encrypted traffic. Results once again confirm findings in Part II - Chapter 1 - Section 1.4 and show unprecedented TLS performance and good indices with IPsec suggesting that the architecture is able to some extent to identify flows in encrypted (VPN) tunnels with far less features.

Furthermore, and since most ML models are used as black-boxes, we provide an explanatory analysis of decisions made by the model as well. Given that the input sequence represent packets, the analysis allows per-packet explanations, enabling the detection/isolation of malicious connections before the flow terminates, therefore a fast deployment of countermeasure and comprehensive (human) investigations.

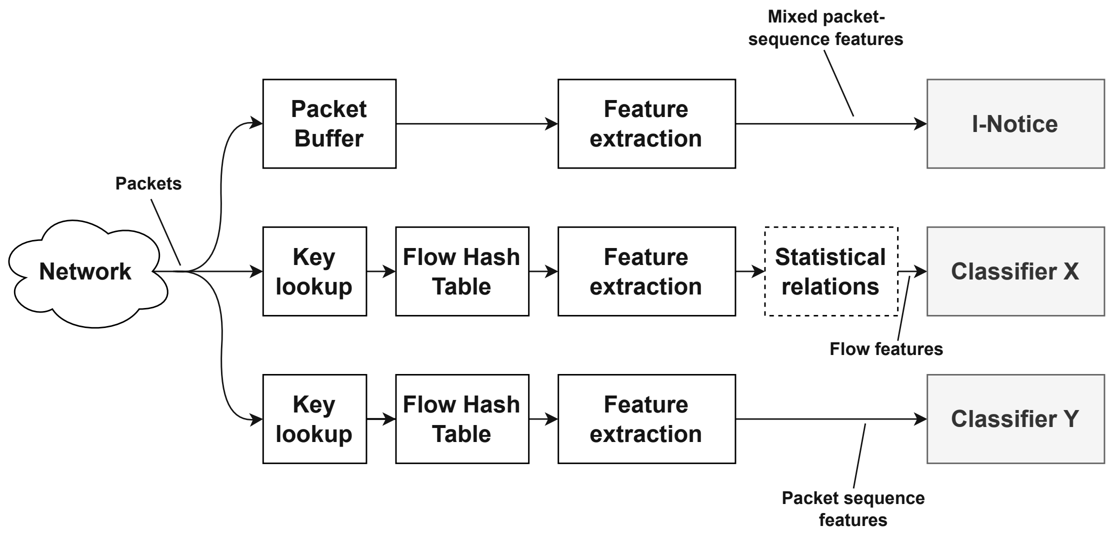
## 2.2 Proposal: I-Notice

The I-Notice framework provides the ability to handle mixed network packets (i.e., packets form different sources to different destinations). Within the following, we address the existing space representations of network traffic used in practice, which are designed for either classifications and/or identification tasks. Afterwards, we delve into each block of the framework, discussing each design decision.

### 2.2.1 Network Traffic

Most learning-based systems for patterns extraction and class identification in communication networks consider traffic from a connection perspective. Namely, packets belonging to the same session/flow are aggregated together after which the required information is extracted. A solution without a flow aggregation has been rarely considered before, as the information in such mixed packet windows is *inhomogeneous* due to the use of different applications and can be extremely *entropic* if many end-hosts are involved. Moreover, despite their remarkable performance, some supervised learning strategies such as the previously used RFs do not support training sequences with mixed labels i.e., an input sample corresponding to multiple attacks. CNNs, by contrast, provide the advantage of detecting artefacts in an incoherent input sequence where the position is irrelevant and therefore offers a solution to our problem. Figure 2.1 shows three different pre-classification architectures of common network classifiers. The architecture in the middle represents the one discussed above (based on flows). The information is represented as flow features (statistical, header values or packet payload). Data collection and construction requires packet aggregation hence, key look-up queries in flow hash tables. The bottom architecture is adapted to sequential algorithms that handle singular packets at each time step (more precisely, a vector of packet features). Yet, such architectures use again the flow concept to aggregate and group packets in order to create input sequences. Finally, the upper architecture represents I-Notice, which requires no aggregation and therefore fewer stages, reflecting less resource allocation.

Although the *multi-key* proposal falls into the category of schemes that require flow aggregation, it is important to note that its goal is distinct and the design was not constrained by computational complexity. As a result, it should not be confused or regarded as a lower-tier solution in this context.



Classifier X: classical flow-based analysis - similar to the *multi-key* proposal
Classifier Y: packet-based analysis per flow
Ours (I-Notice): packet-based analysis per packet

**Figure 2.1:** Common architectures for data preparation. Flow-based aggregation uses a key-lookup in the hash table to aggregate incoming packets, a procedure that needs to be performed for each packet across all previously stored flows.

### Non-encrypted Traffic and TLS

Bearing in mind that the contemporary Internet traffic is mostly encrypted, we also consider what happens if TLS or IPsec encryption are applied. Table 2.1 collects a variety of packet-based attributes used in this work. The baseline feature set contains network and transport header features, which are accessible not only in non-encrypted but also in TLS encrypted traffic therefore, the baseline representation can also be used for this kind of traffic.

### IPsec and VPN

In the case of IPsec encryption; used for instance in VPN tunnels, much less information is available to eavesdroppers. Specifically, in tunnel mode (gateway-to-gateway), a new IP header is prepended to the original packet whereupon all data is encapsulated and encrypted including the original header, making any kind of analysis (except temporal) unfeasible (see Part I - Section 3.4). In transport mode (end-to-end), however, more header fields are accessible. As a result, we focus only on the former case and its implications, the later case is discussed in the next chapter. We use the third feature set in Table 2.1 (IPsec Transport) for this task, which consists of smaller set of packets meta-data features.

| Feature | ID | Baseline | Excluding Transport Ports | IPsec Transport |
|---|---|---|---|---|
| sourceTransportPort | 0 | ● | . | . |
| destinationTransportPort | 1 | ● | . | . |
| packetDirection | 2 | ● | ● | ● |
| ipTotalLength | 3 | ● | ● | ● |
| ipTTL* | 4 | ● | ● | ● |
| ipTOS | 5 | ● | ● | ● |
| tcpFlags | 6-12 | ● | ● | . |
| interPacketTimeSeconds | 13 | ● | ● | ● |
| protocolIdentifier | 14-17 | ● | ● | ● |

*The TTL feature is removed during experimentation due to its high correlation with the label since attackers use the same host all the time (see Part I - Section 3.6).

**Table 2.1:** The three packet feature sets used for sequence representation. We refer to some features using the ID column throughout the paper. Additionally, nominal features are one-hot-encoded.

### 2.2.2 Framework

The framework architecture is shown in Figure 2.2 and consists of five components described as follows:

#### Packet Parser

The packets parser consists of a fixed size buffer that reads packets from the network, followed by a simple feature extractor that reads header values depending on the representation in use (see Table 2.1). Consider in what follows a sequence of $P$ packets and a representation with $F$ features per packet, the data constructed at the output is given by the matrix of size $P \times F$ for each frame. The latter matrix is considered as a data sample and is forwarded to the next component. It is noteworthy that during experimentation, we used static datasets stored in memory and hence, the buffer reads packets from traffic captures (PCAPs) rather than a network interface. However, this does not effect further assumptions. We use go-flows [241] to extract network and transport header values from packets.

#### Latent Space Transformation (1D-CNN)

Each of the previous samples is fed into a 1D-CNN as an image consisting of $P$ pixels and $F$ channels per pixel. Therefore, *every packet is merely a multi-channel pixel.* The internal convolution filters act as information extractors whereby the convolution is maximized if a match occurs. Afterwards, pooling layers pass only relevant information to the next layers. The network is shown in Figure 2.3 (excluding the last four dense layers) and is

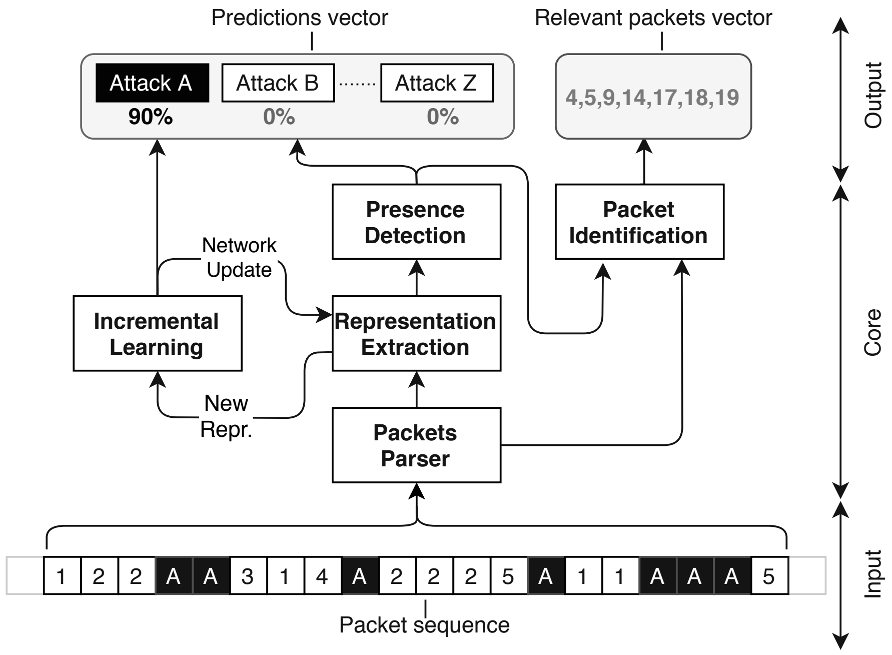


**Figure 2.2:** The I-Notice framework. It consists mainly of three parts: the input represented by a sequence of packets marked with the flow ID. The core, which is the brain of the framework. The output block, which gives predictions and suspicious packet IDs. In this example, the architecture parses the packet sequence from the network and outputs 1) the prediction vector (in this case, the prediction for “Attack A” is maximized) and 2) the sequence of packets causing the activation of the respective label.

based on Lenet-5 used in [145]. After the convolution, we use two pooling layers, the first being the max-pooling layer that captures and filters intra-packets information, and the second being the average-pooling layer that passes the average information to the next component. In addition, after each convolutional layer, we add batch normalization to normalize the sequence values and allow for faster and better training. The number of convolution filters and additional parameters are discussed in Section 2.3.2. Ultimately, the 1D-CNN is used as a feature extractor yielding representations [28]. Namely, it converts a $P \times F$ input matrix into a vector representation of length 128 in the latent space. In other words, the feature extractor acts as a space transformation from a mixed packet sequence into a representative set of values that is assumed to retain maximum traffic information.

### Detection

The last four fully connected layers shown in Figure 2.3 allow to learn the mapping between aforementioned representations and target categories. The last layer (output layer) contains $L$ neurons ($L$ is the number of categories) followed by sigmoid activations

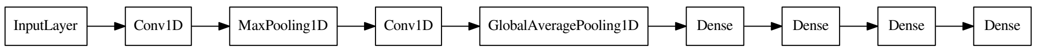


**Figure 2.3:** The deep network used for the classification task. Overall, the network is divided into two blocks: 1) a feature extractor (convolutional layers) that extracts a unique fingerprint from an input sequence and 2) a fully connected layers block which learns the *fingerprint* $\rightarrow$ *prediction* mapping. The architecture is inspired from Lenet-5 [145] and adapted as a 1D-CNN.

to enable multi-label predictions i.e., detecting multiple classes simultaneously. Instead of an additional output dimension, the zero vector ($\underline{\mathbf{0}}$) is assigned as a label for "normal" traffic instances, i.e. all samples that represent benign activity have no output set. The reason behind such a design decision is that normal traffic is scarce and intricate by nature, therefore learning normal traffic patterns is likely to lead to over-fitting due to its wide diversity. Instead, we aim to learn only attack patterns in a semi-supervised setup. During deployment, we assume that the framework yields values close to zero for all kind of benign traffic.

On the other hand, to solve the problem of unbalanced training, we use a weighted loss function that adjusts the loss for each category that is inverse proportional to its occurrence.

### Incremental Learning (Updating Weights and Biases)

Adjusting neural network weights to incorporate new emerging classes without forgetting old ones is not a trivial task. In fact, retraining a neural network causes the parameters to drift significantly and, as a result, the minima of the convex loss function found during the initial training is no longer the global optimum. This phenomenon is described as *Catastrophic Forgetting* [165, 193]. Solutions have been proposed to partially overcome this issue such as EWC [139], LwF [151], iCarl [195], E2E incremental learning [47] and fine tuning [256]. In this work, we opt for a hybrid combination of iCarl and fine tuning due to their high performance and low complexity to add new target classes and, update old ones. We incrementally train only the fully connected layers (detection component) without losing previous knowledge and with minimal change to the original *input* $\rightarrow$ *prediction* function $g(.)$ learned initially. Given an initial trained network $\mathbf{\Omega}_L$ with $L$ classes, a new attack can be learned by retraining the last fully connected layers resulting in a network $\mathbf{\Omega}_{L+1}$. The output layer weights are preserved and an extra (fully connected) neuron is added, bringing the total number of outputs to $L+1$ neurons. Incremental training is conducted using training exemplars, consisting of samples from older classes already trained with, in addition to new samples with new labels. The aim is to minimize the following custom loss function $\ell$ discussed in [195]:

$$\ell(\Omega_{L+1}) = -\sum_{i=1}^{t}[\underbrace{\sum_{y=p}^{t} \delta_{y=y_i} \log g_y(x_i) + \delta_{y\neq y_i} \log(1 - g_y(x_i))}_{\text{classification loss}} \\ + \underbrace{\sum_{y=1}^{p-1} q_i^y \log g_y(x_i) + (1 - q_i^y) \log(1 - g_y(x_i))]}_{\text{distillation loss}} \tag{2.2.1}$$

where:

| | |
|---|---|
| $x, y$ | = training sample, training label |
| $q$ | = network output before incremental training |
| $g(.)$ | = *input* $\longrightarrow$ *prediction* mapping function |
| $\delta$ | = dirac delta function |
| $t$ | = total number of training samples (exemplars) |
| $p$ | = number of previous training samples |

During back-propagation, the loss function is minimized by minimizing (a) the "classification loss" that drives the network to learn new classes and (b) the "distillation loss" that drives the network to maintain old knowledge. Once the fully connected layers are retrained and the loss converges, a short fine tuning phase with fewer epochs and a lower learning rate for all layers (including the convolutional part) is conducted with the aim of enhancing the feature extractor (see [195, 256] for details). Similarly, relearning an existing attack can also be done by updating its samples in the exemplars set and retraining the network as previously discussed without increasing the number of output neurons.

The size of the neural network (number of layers/neurons) plays an important role defining its capacity and, subsequently, the amount of accumulative knowledge that can be learned over time. It is assumed that the initial network is large enough and has a maximum capacity [92] that can handle an increasing number of classes up to a certain limit. To avoid performance deterioration due to the model's capacity insuffisance, we train the network during experimentation with all possible training samples and classes then, we set the obtained evaluation score as an *upper bound*.

### Packets Identification with SHAP

To understand the predictions made by the end-to-end framework, we develop a method utilizing the *DeepExplainer* algorithm of SHAP [160, 222]. Theoretically, SHAP values reflect the marginal contribution of each input feature to the prediction of a model. The values can be obtained by applying the algorithm to a pre-trained model. In our case, the explanations reflect how each packet in the input sequence contributes to the respective prediction. Furthermore, since the convolution causes a dispersing effect (i.e.

the contribution is reflected over multiple packet), we identify a malicious connection by considering only the packet with the maximum accumulative contribution over all features. The identified packet then leads to the end hosts of the connection using IP addresses for instance. To elaborate more, Let $c_{i,j}$ be the shapely value of the i'th packet $\mathbf{p}_i$ and its feature $j$. $k_i$ is the five tuple key given by (here, packet header names are shortened):

$$\begin{aligned} k_i &= \Psi(\mathbf{p}_i) \\ &= \{sIP_{\mathbf{p}_i}, dIP_{\mathbf{p}_i}, sPort_{\mathbf{p}_i}, dPort_{\mathbf{p}_i}, Protocol_{\mathbf{p}_i}\} \end{aligned} \tag{2.2.2}$$

where $\Psi(.)$ is a parser that takes a packet and returns the five tuple key. Let $k_m$ be the key of a malicious packet, then:

$$k_m = \Psi(\arg\max_{i \in \mathcal{P}} \sum_{j=0}^{F-1} c_{i,j}) \tag{2.2.3}$$

Equation (2.2.3) yields the key of a packet in the input sequence that has the maximum contribution over all input features (channels) and therefore, the most relevant for the respective prediction (one of the attack classes in this case). The latter allows the malicious connection to be identified and further isolated by matching all packets with $k_i = k_m$ for $i = 1, ..., P$. It is worth mentioning that Equation (2.2.3) handles only one class explanations. In order to explain multi-class problems; i.e., an input sequence that contains packets from $l$ classes, instead of selecting only the best candidate, we sort packets based on their accumulative contribution, select the top distinct $l$ packets, retrieve their keys and identify each communication flows.

### 2.2.3 Heuristic Complexity

The complexity of I-Notice is accumulated over four computation blocks:

1. *Packet buffering.* The first task is to capture packet sequences (e.g. 20 packet each window) and extract header features as shown in Table 2.1. This can be achieved using a network interface, a buffer and requires no flow aggregation or complex computations.

2. *Forward propagation.* Generating predictions is a lightweight procedure. It consists of multiple vector/matrix multiplications and can be executed on low-end devices.

3. *Initial training.* The complete network is trained initially with available training samples in a semi-supervised manner. This phase requires more resources nonetheless, it is typically executed on a training server before actual deployment.

4. *Periodic incremental training.* When new classes emerge, the framework can be retrained only for a few epochs hence requiring less time and allowing for training on special low-end hardware.

## 2.3 Evaluation

We examine each component in the framework through various tests. In what follows we describe the experimental data and the training procedure. Furthermore, we collect the set of experiment scripts in a repository[1] to encourage reproducibility.

### 2.3.1 Datasets

Two intrusion detection datasets in addition to a backbone link network trace (described in what follows) are used for training and testing. Experiments are divided based on their purpose into three categories:

- Learning representations (latent space construction). We use weekly traces of the Measurement and Analysis on the WIDE Internet (MAWI) data [225]. The latter are network traffic captures extracted from a backbone link. The data is not labeled nor cleaned and therefore, it reflects the perfect representation of the data shape for the initial training of the feature extractor. Such highly diverse data allows the 1D-CNN to learn a variety of representations in an unsupervised manner (see Section 2.2.2).
- Proof of concept. We use the CICIDS2017 dataset [220] in all other experiments (i.e., classification, incremental learning and model explainability). In addition to benign traffic profiles, the dataset offers a set of 14 network attacks.
- Validation. In order to validate classification performance and generalize conclusions, we use the UNSW-NB15 dataset [176] in addition to the CICIDS2017 dataset. UNSW-NB15 offers a set of 9 network attacks in addition to modern normal traffic.

### 2.3.2 Training

The training procedure is split into three phases: (1) Learning Representations, (2) Ground Classification and (3) Incremental Learning.

#### Learning Representations

To compulse the feature extractor discerning traffic patterns, we use a backbone-link dataset (MAWI) to train a convolutional auto-encoder shown in Figure 2.4. The encoder part (stages 1 and 2) represent the feature extractor used in I-Notice and discussed in Section 2.2.2. The decoder is a symmetrical architecture. We create a bottleneck layer at which the latent space is constructed with the goal keeping the *input* $\rightarrow$ *output* similarity at its maximum. Furthermore, we use the model transferability theory to learn inclusive patterns of network traffic from MAWI, then use the trained encoder as a feature extractor for better generalization. We Mean Squared Error (MSE) as loss function and

[1] `github.com/CN-TU/i-notice`

back-propagate the gradients using the an *Adadelta* optimizer [259] ($lr = 0.001$, $\rho = 0.95$, $\epsilon = 10^{-7}$) as this combination was shown to converge rapidly [259] in most applications.

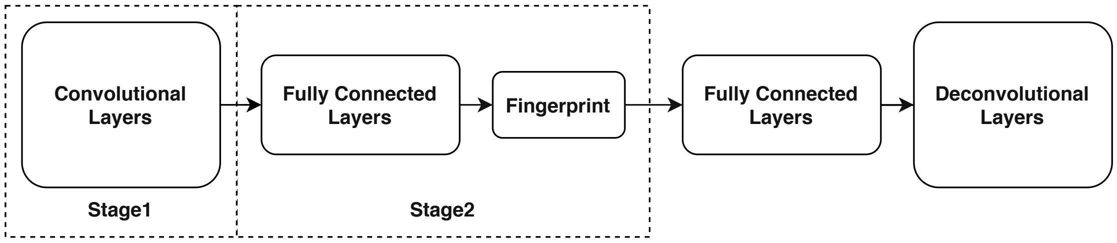


**Figure 2.4:** The architecture of the convolutional auto-encoder used to pre-train the convolutional layers of Figure 2.3. The encoder and decoder architectures are symmetrical and the bottleneck layer yields a fingerprinting vector of length 128.

### Detection and Classification

Once the auto-encoder loss converges, we purge and install the encoder block in the complete architecture then fine tune it using CICIDS2017 and UNSW-NB15 in a supervised manner. The datasets are divided into training ($\frac{2}{3}$) and testing ($\frac{1}{3}$) portions. In addition to various input sequence lengths (20, 100 and 1000), the output layer is substituted to create two models described as follows:

- **Binary classification.** The output layer consists of only two neurons, predicting either benign or malicious traffic. We use the soft-max activation function:

$$O_i = \frac{e^{z_i}}{\sum_{j=1}^{2} e^{z_j}} \tag{2.3.1}$$

  To obtain complementary probabilities for each binary class. We train with a categorical cross-entropy loss and *Adam* optimizer [138] ($lr = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$). To inspect the effect of encryption on performance, we also alter the input layer to handle all representations mentioned in Table 2.1.

- **Multiclass classification.** In this case, the output layer consists of $n$ output neurons standing for all attacks in the dataset (excluding normal traffic which is represented by a zero vector). We train the network using a binary cross-entropy loss to allow for multi-label predictions (each output is penalized separately), a sigmoid activation is henceforth used to allow for multiple active logits:

$$O = \frac{e^z}{e^z + 1} \tag{2.3.2}$$

  We optimize the loss using *Adam* optimizer with the same parameterization as before.

| Dataset | Variant | Sequence Length | Lower Bound | Baseline | Without Ports | IPsec Transport |
|---|---|---|---|---|---|---|
| CICIDS2017 | Multiclass | 20 | 0.907 | **0.985** | 0.982 | 0.980 |
| | | 100 | 0.891 | 0.982 | 0.978 | 0.974 |
| | | 1000 | 0.852 | 0.964 | 0.960 | **0.940** |
| | Binary | 20 | 0.907 | **0.991** | 0.988 | 0.985 |
| | | 100 | 0.891 | 0.984 | 0.980 | 0.975 |
| | | 1000 | 0.852 | 0.963 | **0.934** | 0.941 |
| UNSW-NB15 | Multiclass | 20 | 0.904 | **0.936** | 0.931 | 0.930 |
| | | 100 | 0.726 | 0.807 | 0.795 | 0.740 |
| | | 1000 | 0.503 | 0.546 | 0.547 | **0.538** |
| | Binary | 20 | 0.904 | **0.980** | 0.977 | 0.962 |
| | | 100 | 0.726 | 0.950 | 0.941 | 0.930 |
| | | 1000 | 0.503 | 0.933 | 0.897 | **0.667** |

**Table 2.2:** Prediction accuracy based on different sets of parameters. Both best and worst performance are highlighted.

Both variants are trained with a batch size of 32 and a maximum of 128 epochs. However, we monitor the validation loss and stop training if no improvement is observed to avoid over-fitting. By counting 2 variants, 3 sequence lengths, 3 feature sets and 2 datasets, we train 36 different models and summarize the results in Table 2.2.

### Incremental Learning

As discussed before, incremental learning enables adding knowledge to our initial model $\Omega_L$ later after deployment. We employ the pre-trained architecture described above and add classes incrementally. To achieve so, we store a set of training exemplars in memory for classes that are already learned and are correctly classified by the current model $\Omega_L$. Each time a new attack emerges and is set to be learned, we collect representative training samples, merge them with the existing set of exemplars and initiate training using the loss function described in Equation (2.2.1). The exemplars of an attack $A$ are chosen using *herding* (described in [195]) to minimize the subset variance i.e., we compute the mean vector of all training samples of an attack $A$ ($\mu_A$), we then select those samples from the training set closest to the mean and finally add them to the exemplars set. The network is then trained using the new exemplars set and fine-tuned for 3 epochs which results in the model $\Omega_{L+1}$.

Performance are shown in Figure 2.5. In addition, we show the upper and lower bounds values that represent: training with all classes at the same time (see Section 2.3.2) and random majority-class guessing, respectively. Intuitively, if the memory requirements are ignored and number of exemplars is not bounded, the classifier's performance converges

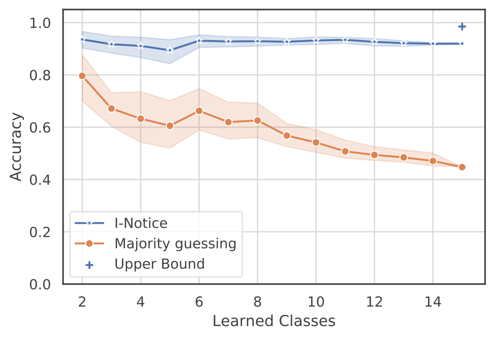


**Figure 2.5:** Averaged accuracy of an incrementally trained model. Overall, the proposed model performs significantly better than majority class guessing and almost reaches the upper bound.

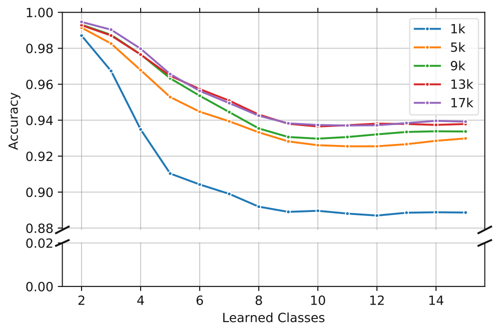


**Figure 2.6:** Accuracy achieved in terms of number of exemplars during the incremental training.

to the upper bound. However, due to limited memory assumptions, we do not store all previous training examples but instead we use herding. Figure 2.6 shows the effect of the number of exemplars on the incremental accuracy. The x-axis represents the number of classes learned incrementally and the y-axis shows the accuracy achieved for each number of stored exemplars per-class. Obviously, storing more exemplars yields better accuracy.

## 2.4 Results and Discussion

We benchmark I-Notice and its different components by evaluating the initial classification performance and the incremental training procedure. Next, we analyze the explainability plots to build an understanding about the outputs made by the network and whether they are reasonable from a network security perspective. Finally, we discuss possible adversarial attacks against I-Notice based on probabilistic analysis.

### 2.4.1 Proof of Concept

Table 2.2 summarizes the classification accuracy of 3 feature vectors, 2 datasets, 2 classification variants and 3 sequence lengths. The lower bound represents the performance of a *dummy* classifier, which always predicts the majority class.

The best-performing combination for CICIDS2017 is the variant with: {sequences of 20 packets each, binary prediction and baseline representation}. The later observation also applies to UNSW-NB15. In the same way the combination: {sequences of 1000 packets each, multiclass prediction and the IPsec representation} performs the worst. This is expected since the IPsec representation carries less information and a long input sequence reflects higher entropy which makes the learning of the feature maps at the convolutional

| **Dataset** | **Metric** | **LSTM** [103] | **RF**[219] | **DNN-p*** | **Ours** |
|---|---|---|---|---|---|
| CICIDS2017 | Acc. | 0.998 | na | 0.924 | 0.990 |
| | Prec. | 0.999 | 0.980 | 0.930 | 0.981 |
| | Rec. | 0.992 | 0.970 | 0.921 | 0.909 |
| | F1. | 0.995 | 0.970 | 0.922 | 0.943 |

| **Dataset** | **Metric** | **DNN** [134] | **RF** [172] | **DNN-p*** | **Ours** |
|---|---|---|---|---|---|
| UNSW-NB15 | Acc. | na | 0.989 | 0.908 | 0.980 |
| | Prec. | 0.891 | 0.853 | 0.822 | 0.831 |
| | Rec. | 0.632 | 0.847 | 0.837 | 0.807 |
| | F1. | 0.908 | 0.843 | 0.825 | 0.815 |

*DNN-p: Deep feed-forward neural network handling single packets.

**Table 2.3:** Comparative analysis for both IDS datasets.

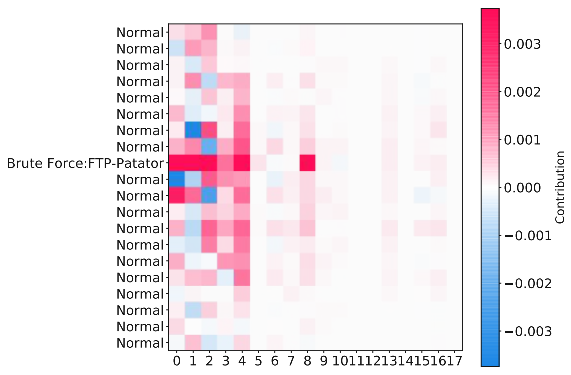
layers difficult. This can be resolved by providing significantly more training samples. In any case, I-Notice still performs better than the lower bound.

In contrast to related work, I-Notice uses mixed packet sequences as input. Therefore the comparison of detection results with flow-based methods is not straightforward. However, we still show the classification scores of recent work for both datasets in Table 2.3. It is worth mentioning that although other competitors have higher scores, the task solved by I-Notice is more sophisticated i.e., considering sequences of packets from mixed flows rather than flow-based aggregated packets.

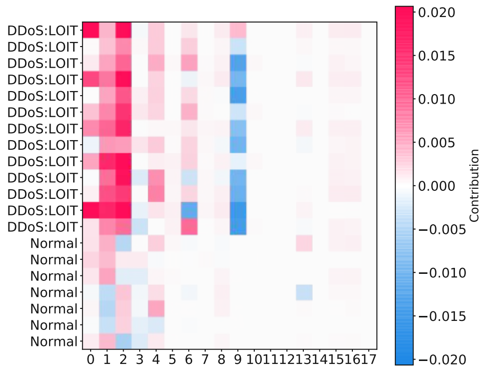
### 2.4.2 Model Interpretability

To better understand the approximation function $g(.)$ created after training the model $\Omega_{15}$, we use SHAP and dissolve the neural network with 20 packet input sequences. Three samples are shown in Figure 2.7. The horizontal axis represents packet features (as input channels) and the vertical axis represents a random sequence of packets. The color map shows the contribution of each {packet, feature} pair to the output label of the respective attack (i.e., FTP-Patator, GoldenEye and LOIT). The first scenario shows a single malicious packet, the second when half of the packets are malicious, and the last one when the entire sequence is malicious. The plots show higher contribution at malicious packet positions however, a minimal contribution is distributed and fades across other neighboring {packet, feature} pairs mainly because of the convolution. Equation (2.2.3) therefore always succeeds at identifying one of the malicious packets and thus in practice, the abnormal communication flow can be isolated by observing the SHAP values.

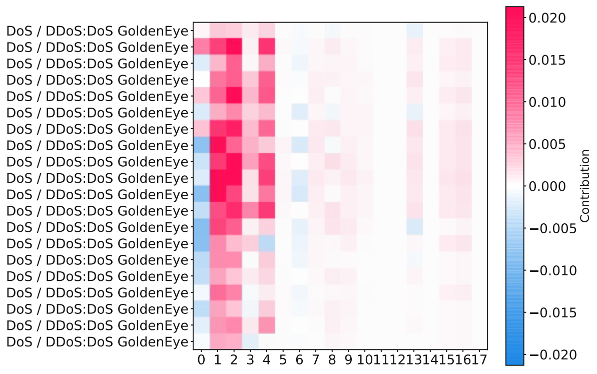
At a finer granularity, Figure 2.8 shows the average SHAP value per attack and feature over all test samples. For each attack, the contribution of all features sums up to ~100%.



**(a)** Only one malicious packet.



**(b)** More than half of the packets are malicious.



**(c)** All packets are malicious.

**Figure 2.7:** SHAP values of three randomly chosen malicious windows. The horizontal axis represents the packet features (channels, see Table 2.1) and the vertical axis represent the ground truth label of each packet in the window. The colors represent the contribution of each (packet, feature) pair to the corresponding attack prediction.

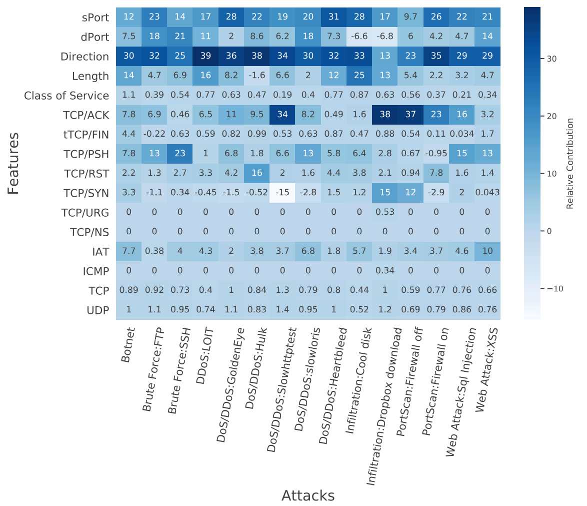
In Figure 2.8a we consider the baseline representation in which we see that, depending on the class, the model treats features differently. We mark three main observations:

- **Packet direction.** The packet direction (server → client or client → server) is relevant for almost every category. Specifically, we see that for DOS attacks, for example, where the number of packets going in one direction (attacker → victim) is higher and therefore, the direction is relevant. In other scenarios e.g. Infiltration/download, the direction is less relevant as the connection is symmetrical due to acknowledgments after each data transfer.
- **Transport ports.** Communication ports have been already shown to be relevant for classification tasks [85]. Results suggest that this continues to be the case. Source or destination port numbers are still relevant to identify standard services and protocols such as web browsing and subsequently web-related attacks (e.g. SQL injections or XSS)
- **TCP flags.** The set of transport layer flags used in the baseline representation show different levels of relevancy depending on the target class. File transfer (Infiltration) or port scanning highly depend on the TCP/ACK flag, intuitively because an increase in the count of this flag can reflect the reaction of a victim under attack. TCP/PSH flag is relevant for SSH brute forcing attacks because the latter is used to push data immediately to higher layers without delay which subsequently allows for a faster brute forcing. TCP/RST flag is relevant for detecting Hulk DDOS attacks because it give a hint that the server resources are saturated.

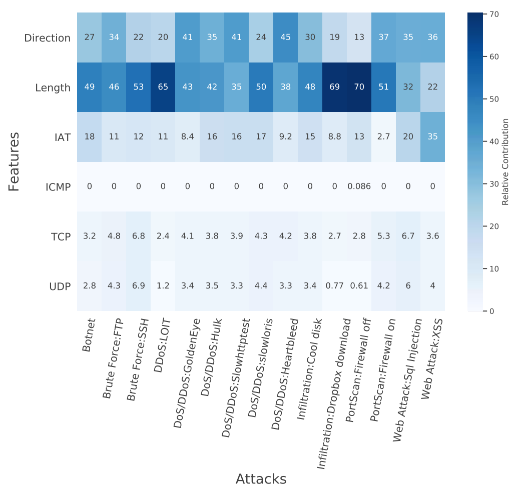
It is clear that the classifier has learned the expected characteristics of the traffic making the predictions no longer blind. In fact, I-Notice provides immediate insight into the cause of the alarms.

We further interpret predictions in the case of the IPsec representation, whereby the input space is much smaller. Figure 2.8b shows a similar distribution to the one explained above but this time with only metadata features. In a similar manner, we mark three main observations:

- **Packet direction.** The packet direction is still relevant for the same reason as before, even if tunneled through VPN tunnels.
- **Packet length.** The length of the packet has become the most relevant feature. Out of all remaining features, this is the most likely one to differentiate between packets (and contain useful information). For some attacks, such as port scanning, where the size of the packet is usually the same, the packet size feature has a high contribution. For other classes, such as web attacks, the payload may contain payloads with different sizes and therefore, the length of the packet does not contribute much.



**(a)** Baseline representation with 20 packets at the input.



**(b)** IPsec representation with 20 packets at the input.

**Figure 2.8:** The average SHAP values over all testing samples per features and attack family. Darker areas represent a higher contribution to the respective category.

- **Protocol.** Another feature that has become relevant is the packet protocol. The latter also contributes useful information to distinguish between classes. Whether it's TCP and UDP traffic, these two share the majority of the contribution, whereas Internet Control Message Protocol (ICMP) is not relevant since most attacks use either of the first two protocols.

One should keep in mind that easily interpretable models reflect an easily fooled model (by generating adversarial samples). This can be done for the IPsec case by altering the payload using padding or encapsulating packets to change the protocol number. We address this problem in Section 2.4.4.

### 2.4.3 Incremental Learning

Figure 2.5 shows the average accuracy of an incrementally trained classifier using a maximum of 15 classes and 5000 exemplars per class (refer to Section 2.2.2). The upper and lower bounds representing the baseline classifier trained with all classes simultaneously (Section 2.3.2) and the majority guessing classifier are shown for comparison. To understand how the order of training affects the overall accuracy, we repeat experiments 100 times with random class order and additionally show the standard deviation of the accuracy. Clearly, performance decreases each time an additional class has to be learned. This is because weights are updated in order to optimize for the second loss function while the first one may not be at the same minimum anymore [139]. However, we observe that the final score (when reaching the 15th class) is closer to the upper bound indicating that the incremental learning is functional. Learning attack patterns incrementally is a very important feature of any modern IDS as real attacks are discovered on a daily basis and defense mechanism should react immediately. I-Notice offers a rather straightforward, initial solution to this challenge.

When allocating memory storage for I-Notice, the size of the exemplars set enters into play. In Figure 2.6, we show how the number of exemplars affects the classification accuracy. To achieve so, we use the best performing class order from the previous test and vary the number of exemplars that we keep in memory (1k, 5k, 9k, 13k and 17k). Observations show that when storing 5k exemplars or more the accuracy does not improve much, but decreases significantly if we reduce the number of exemplars. We therefore opt for 5k exemplars per class. This number might change in practice and is highly dependent on the quality of training data in addition to the attack characteristics.

### 2.4.4 Adversarial Attacks and limitations

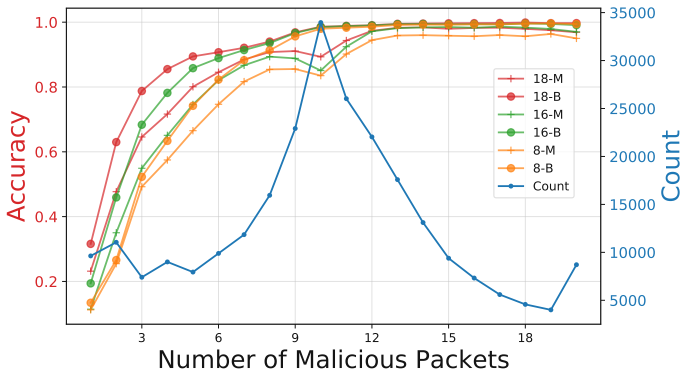
In this part we discuss possible white-box adversarial attacks that can drive the model to give wrong predictions given the set of feature relevancy values. In Figure 2.9, the achieved accuracy in relation to the amount of malicious packets per input sequence is shown on the left vertical axis. Additionally, we show the count of each value across the entire dataset on the right vertical axis e.g., in the first figure we observe ~35k sequences with 10 malicious packets out of 20 total packets. Benign sequences are not counted here due to their high occurrence as most network traffic is normal. The accuracy increases proportionally with the number of malicious packets in a sequence i.e., the more malicious packets there is in a sequence, the higher the probability it get's correctly classified. In the third case (1000 packets per sequence), the accuracy decreases for sequences with more than ~750 malicious packets because there are very few such samples reflecting that statistics are not representative.

Based on previous observations, we suggest three possible scenarios in which adversarial samples can be generated by an attacker:

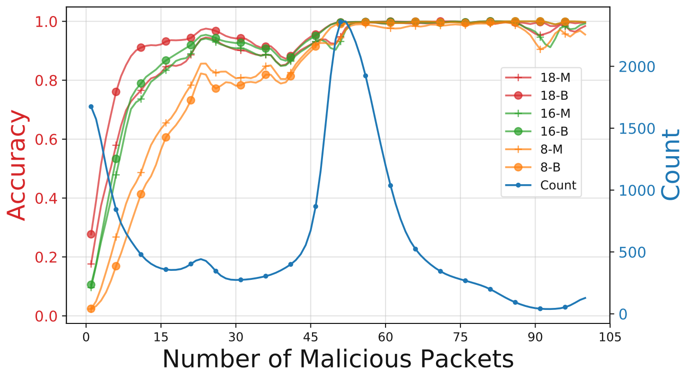
- **Packet length manipulation for IPsec.** As discussed in Section 2.4.2, the length of a packet is a major contributor to predictions for the IPsec representation. Altering the length by padding bytes to the payload may therefore have an unexpected effect on the model's output. In ESP mode [133], this is feasible by using the padding field which may contain randomness. The former manipulation is difficult to detect and still a challenge for most IDSs.

- **Packet sparsity.** As shown in Figure 2.9, sequence samples with few malicious packets are barely correctly detected. As a result, an attacker can hide its malicious activity and therefore the actual packets by sending them with random delays, i.e. increasing the inter-arival time and sending fewer packets per sequence. This approach has limitations and is unrealistic for certain attacks e.g., DOS where the burst cannot be circumvented otherwise the attack would fail.

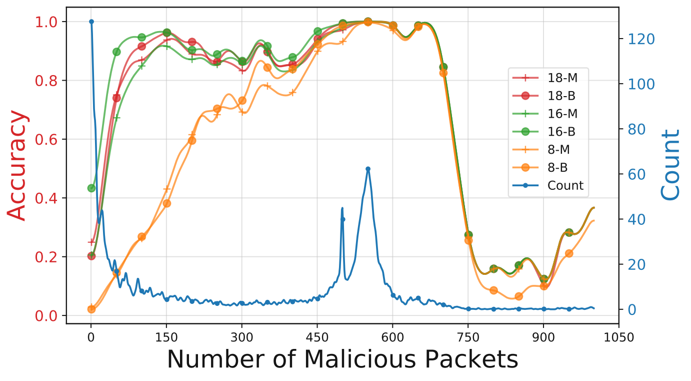
18: Baseline, 16: Excluding Transport Ports, 8: IPsec Transport, M: Multiclass, B: Binary.



**(a)** 20 packet sequences.



**(b)** 100 packet sequences.



**(c)** 1000 packet sequences.

**Figure 2.9:** Prediction accuracy (left y-axis) in terms of how many malicious packets a sequence contain (x-axis) for the CICIDS2017 dataset.

- **TCP flag crafting.** If additional TCP flags are manually set ensuring that neither the communication nor the attack are affected, the added noise may introduce confusion for the classifier and therefore reduce its confidence e.g. the attacker sends the first TCP packet with a TCP-RST flag set instead of a TCP-SYN thus, confusion the classifier without causing problems from a connection perspective.

All of the previous attacks are applicable only if the attacker has full knowledge not only of the classifier's architecture but also the weights which allows to reconstruct the contribution figures however, this is usually not trivial.

We emphasize that many work use the TTL feature too for such applications but we removed it during initial investigations analysis due to its high correlation with the attacker's host and, subsequently, the prediction. This issue was raised by authors in [21] and how it is possible to break a fully functional IDS only by manipulating this feature (further discussed in Part I - Section 3.6).

## 2.5 Conclusion

Although the usage of ML for cyber-security and attack detection applications has been extensively explored and various methodologies have been proposed in the past, many approaches ignored important aspects of practical implementations. The complexity of the system, its ability to adapt over time and its capabilities of handling encrypted traffic are crucial requirements when developing a modern IDS architecture that has to be undoubtedly satisfied.

In this chapter, we propose I-Notice, a new anomaly detection framework with an unconventional representation of heterogeneous packet sequences for network traffic. We show that I-Notice obtains satisfactory performance given the ambitious problem that needs to be resolved compared to the *multi-key* approach discussed in the previous chapter. I-Notice can handle unencrypted traffic, TLS and partially IPsec (transport mode) encrypted traffic. We further show that SHAP values of a pretrained model give expected explanations of predictions, which in turn validate the performance and enable a better understanding and faster response from network administrators. We study the possibility of incrementally learning new attack classes as this represents an important aspect of network traffic because new attacks are triggered on a daily basis. Experiments show that the architecture can be retrained without largely losing old knowledge and maintains a stable accuracy over time. Overall, experiments show that it is possible to detect singular anomalous packets with satisfactory performance without the use of network flow aggregation.

CHAPTER 3

# Flow Counting with Minimal-information Feature Vectors

***Notice of adoption from previous publications - 5***
*Parts of this section's content have been published in the following paper:*
*Fares Meghdouri, Félix Iglesias Vázquez, and Tanja Zseby. Shedding light in the tunnel: Counting flows in encrypted network traffic. In* 2021 International Conference on Data Mining Workshops (ICDMW)*, pages 798–804. IEEE, 2021*

In the previous two chapters, we have explored how the design of the input space (traffic representation) can partially aid solving the problem of detecting network anomalies. In this chapter, we focus on creating a traffic representation for counting network flows in encrypted tunnels.

## 3.1 Introduction

One of the preliminary tasks before designing an intrusion detection, traffic monitoring, or quality control system consists on gathering statistics about traffic passing through a specific observation point and disclose the number of active network flows. Having access to this information allow estimating the number of connections, hosts, or the network size in general [25, 107]. However, the latter is difficult due to the fact that encrypted traffic is often routed through network tunnels in which the original IP packet is encapsulated with a pre-header at aggregation points. For instance, when using IPsec in tunnel gateway-to-gateway mode, all traffic leaving a local network has the IP address of the network gateway, making it difficult to separate or count the number of active flows once in the IPsec tunnel.

Previous research indicates that the flow count is required for a variety of applications [136, 110, 149, 217]. Some examples are:

- Monitoring resource usage for network management.
- Detection of specific types of attacks, e.g., Distributed Denial of Service (DDoS); and anomalies in periodic/predefined communication patterns.
- Checking the state of sensors or IoT devices, for example in smart grids or smart production environments where each devices maps into one flow.
- De-anonymizing traffic to determine the number of communication parties.

In this chapter we propose simultaneously a representation and an architecture that uses LSTM networks [109] to solve the problem of counting concurrent flows in network tunnels. The proposed representation uses packet features that are accessible even when traffic is encrypted, namely: *packet size* (PktSize/ipTotalLength), *inter-arrival time* (IAT/interPacketTimeMilliSeconds) between packets, and *packet direction* (Direction/packetDirection). Subsequently, our approach is able to estimate the number of concurrent flows in VPNs tunnels based on the LSTM attention mechanism.

We summarize our contributions as follows:

- We propose a lightweight representation combined with an architecture capable of accurately estimating the number of flows in encrypted tunnels.
- We analyze data from two consecutive years (MAWI datasets from 2020 and 2021) and show that many traffic characteristics did not significantly change over this time frame. Subsequent experiments show that, the proposed technique is therefore time-transferable in our settings (only requiring eventual fine tuning).
- We show that our proposal is robust against possible adversarial attacks that aim at influencing model predictions by manipulating certain traffic features.

## 3.2 Proposal: LSTM Networks for Flow Counting

As discussed in the previous section and Part I/Section 3.4, traffic encryption makes traditional data extraction methods hardly feasible (including some methods discussed in Chapter 1 and Chapter 2). Here we focus on the case of IPsec tunnels that is increasingly common in Internet traffic. All packets from a network are aggregated at the gateway, typically a router, which encapsulates, adds a message authentication code and encrypts all packets before sending them over the Internet to the receiving gateway, thereby constructing a tunnel. Since packets inside the tunnel have the same origin and destination header, determining how many flows are transported cannot be done using traditional flow-key hash methods. Hence, we are forced to use the external network information that is still available regardless encryption for this task namely: packet size, packet direction, and times between consecutive packets. The following (reasonable) requirements are assumed in our proposal:

1. Gateways introduce minimal queuing delay. Otherwise, inter-arrival times would be affected.
2. Gateways encapsulate packets with fixed pre- and post-headers (either ESP or AH), increasing the size of all packets in the same way.
3. Gateways do not introduce application-based payload padding.

Note that when dealing with the inter-arrival time between packets, we in fact consider two attributes: the first represents the time between packets regardless of their direction and a second one that only represents the time between packets of the same direction i.e., the time to the last packet with the same direction. The previous reasoning is derived from the observation that packets of the same direction reflect typically the behavior of a single source, whereas when considering both directions more uncertainty is introduced because two consecutive packets of different directions represent the interaction of multiple sources and may reflect an arbitrary inter-arrival time (e.g. server/client response times).

### 3.2.1 Proposed Solution

The complexity of counting flows is highly dependent on the number of multiplexed flows, the nature of each flow, and the algorithm used to aggregate the flow (i.e. software running on gateways). Furthermore, because packets are generated based on user interaction, the resulting traffic usually does not exhibit stationary time patterns. We summarize the most important symbols used throughout the paper in Table 3.1.

| Symbol | Description |
|---|---|
| N | number of packets in a sequence |
| M | number of sequences in a dataset |
| $\Delta t$ | time between two consecutive packets |
| $\mathbf{s}$ | one sequence of packets |
| $\Gamma$ | set of observed $\Delta t$'s |
| $y_{i,j}$ | actual count for the $i^{th}$ sequence at $j^{th}$ packet |
| $\widehat{y}_{i,j}$ | LSTM output for the $i^{th}$ sequence at $j^{th}$ packet |
| $\Delta\widehat{y}_{i,j}$ | increment in the LSTM output |

**Table 3.1:** Notation.

The example depicted in the upper part of Figure 3.1 shows two hosts sending packets on a regular basis and then aggregated in a tunnel. This example could represent two regular sensor reporting in a M2M setting. Sensors send packets with fixed time periods of $\Delta t_1$ and $\Delta t_2$ respectively. Since they are mixed within the tunnel, observing the final transmission does not reveal information about the number of sensors or the individual periods. However, it is possible to guess flows whenever $\Delta t_1$ and $\Delta t_2$ are known a-priori. Let **s** be a sequence of $N$ packets in a tunnel, we can iterate over **s** and store the observed delays in a finite set $\Gamma$. If we increase the number of counted flows only when spotting delays that are not already in $\Gamma$, we are able to roughly guess the number of flows, in this case, it would be 2. This manual approach would work only if the assumption that flows exhibit some periodicity ($\Delta t$ are known and constant) is met.

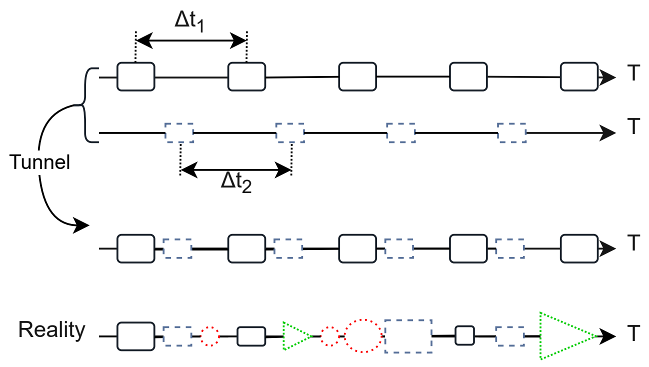


**Figure 3.1:** An example of a simple tunnel with two periodic transmissions (upper part) and a realistic Internet tunnel (bottom part). Here different shape and color means different application and different size means different packet size.

Such rule-based approaches (*if-then-else*) could be enriched with information concerning packet size and packet direction. However, rule-inference-based systems show some drawbacks [99, 30] too:

- Input data must be cautiously cleaned because noise can have a significant impact on a *yes/no* rule.
- There is no way to prove the absolute correctness of a rule inferred from previous observations, even if they are similar in the future.
- With the accumulation of nested rules, their order becomes important causing the generation of sub-optimal rule trees and branches, the degradation of the learning process itself, and an increasing opacity in the model (understood as a set of rules) interpretation.

The complexity of a rule-based system can quickly become unmanageable, requiring automated models built with regression or rule-induction methods. We opt for an advanced model-learning attention technique that (a) captures patterns instead of rules, and (b) stores and uses only what is required from past events, LSTM cells are therefore a perfect fit to solve this problem. After enough training, the model is expected to accurately guessing the number of flows in a network tunnel given a more complex setup.

### 3.2.2 Managing Complexity and Memory

Similarly to I-Notice, we treat packet sequences as multidimensional time series and assume the existence of temporal dependencies between them. RNNs architectures have been extensively used to solve this kind of problems. Such models use current data samples in addition to old information acquired from previous samples to construct valid predictions at each time step. However, RNNs are well known to suffer from the vanishing gradients for long sequences causing the model to quickly forget past knowledge [185]. In order to overcome this limitation, we make use of a a similar but more elaborated architecture that solves the previous problem: LSTM.

LSTM cells are a type of RNNs that have an enriched internal architecture. In addition to the recurrence, LSTM units contain more inner gates that allow to efficiently keeping past knowledge and use it in combination with future inputs. RNN units usually contain only an input gate that merges together the previous hidden state of the unit with the new input and an output gate. Instead, LSTM units introduces the new concept of the forget gate and the cell state. The forget gate decides whether to keep new knowledge or to "forget" it and the cell state is a combination of the previous cell state, the new input and the forget gate state. The idea behind is that, over time, the unit learns which information to remember and which to forget, this is intuitively similar to what happens in NLP applications as discussed in Part I/Section 2.3.2. We configure our architecture so that each time step represents a new captured packet and the multidimensional input vector reflects the four features discussed above, namely: PktSize, Direction, mixed IAT, and same-direction IAT. Therefore, we construct sequences of length $N$ and dimension 4. The cell output $\widehat{y}_{i,j}$ is the estimated number of flows up to the current time step (or packet) $j$ for a given sequence $i$.

It is noteworthy that our proposal uses sequences of fixed length. Such sequences represent sets of $N$ packets taken from a FIFO buffer rather than a single infinite sequence[1]. A defined sequence length is required to prevent error propagation for long sequences if the network prediction begins to drift from the actual number of flows. By having a short sequence length, the cell state is flushed at the beginning of a new sequence and the error is hence minimized. However, extremely short sequences suffer from the lack of necessary information and lead to poor predictions.

[1]In the case of an infinite sequence, the cell state is never flushed and predictions are obtained after each time step continuously.

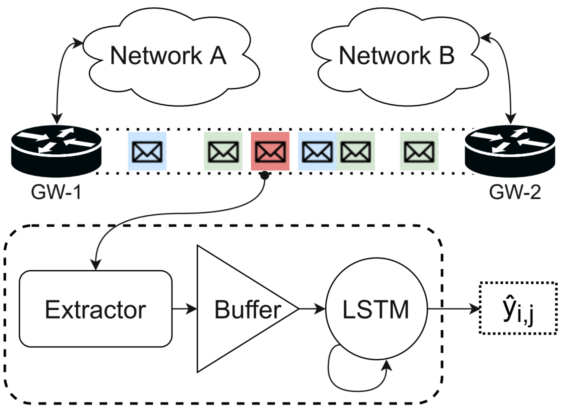


**Figure 3.2:** LSTM-based solution to count flows in a tunnel. The network is fed only with available packet metadata.

The architecture shown in Figure 3.2 depicts two E2E flows being routed through a tunnel with GW1 and GW2 acting as VPN aggregation gateways. The counting chain is divided into several stages: (1) parsing network packets, (2) extracting and processing of required features, (3) using a buffer to store fixed-length sequences of feature vectors, and (4) step-wise feeding of sequences into the LSTM model. The first three stages which represent data preparation are similar to the architecture proposed in I-Notice, except for the new representation developed here. The LSTM updates its state and predicts the number of flows ($\widehat{y}_{i,j}$) at each time step of the current observation window. The proposed approach is able to count the number of new flows incrementally, i.e., we predict the start of a new flow; but keeping track of active flows is hardly feasible since the termination of a flow is often defined by specific transport layer flags that are not accessible with encryption. Estimating the number of active flows would require flow-duration estimation, which is not analyzed here.

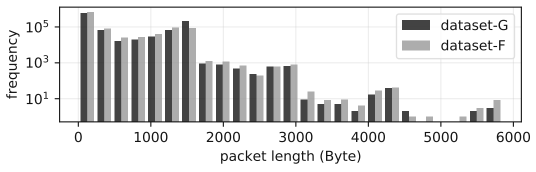
## 3.3 Evaluation

In this section we discuss the methodology used for the evaluation and relevant details related to the reproduciblity of our results. The source code, data, and pre-trained models are available online[2].

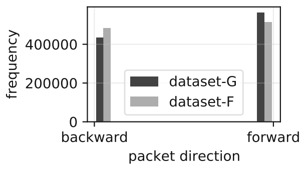
### 3.3.1 Data

For our experiments we use real traces captured by the MAWI[3] working group at backbone links. Data collected during June 2020 (Samplepoint-G, average rate: 2.3Gbps with ~465 million packets) is used for training (60%), validation (20%) and testing (20% -

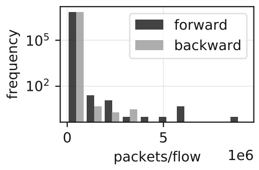
[2] github.com/CN-TU/network-flow-counting
[3] mawi.wide.ad.jp/mawi



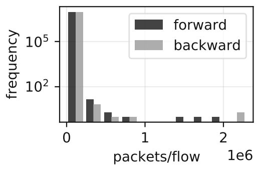
**(a)** Packet length.



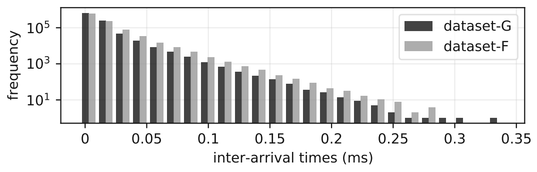
**(b)** Packet direction.



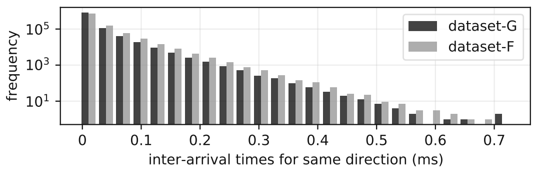
**(c)** Dataset-G.



**(d)** Dataset-F.



**(e)** Mixed IATs.



**(f)** Separate IATs.

**Figure 3.3:** Histograms showing the distributions of relevant features in the datasets used for evaluation

Test 1). We then use a second capture from February 2021 (Samplepoint-F, average rate: 420Mbps with ~83 million packets) for a second test (100% - Test 2) in order to check if a trained model is time-transferable and can be used with network traffic observed several months later at a different sampling point. We show statistics related to traffic characteristics of both sampling points in Figure 3.3. Direction is *forward* when the packet is sent by the device that started the flow; otherwise, "packet direction"‘ is *backward*. TCP is obviously the most frequently used protocol in both traces. We show the distribution of various protocols in Table 3.2.

| **Dataset** | **TCP** | **UDP** | **ICMP** | **Others** |
|---|---|---|---|---|
| Samplepoint-G | 76.7% | 15.8% | 01.9% | 05.6% |
| Samplepoint-F | 61.2% | 29.6% | 00.6% | 08.6% |

**Table 3.2:** Protocol distribution in MAWI network traces.

### 3.3.2 Configuration of the Learning Scheme

Our neural network consists of a three-cell stacked LSTM that takes sequences of a length $N$. In our experiments, we test different sequence lengths, $N = \{50, 100, 500, 1000\}$.

We empirically set 128 neurons for each intermediate layer with *Leaky ReLU* [162] activations and a *sigmoid* activation at the output layer. Dropout is used at each layer with $r = 0.2$ to reduce over-fitting [227]. During training, the Mean Absolute Error (MAE) is used to compute the loss at each step of the sequence. The reason behind using the MAE is its robustness against outliers compared to a simple MSE which usually aggravate larger losses (caused by outliers). An *Adam* optimizer [138] is used loss optimization/back-propagation.

To deal with the long tailed distribution of packet sizes and inter-arrival times (see Figure 3.3a, Figure 3.3e and Figure 3.3f), we apply a log transform (Equation (3.3.1)), which focus on orders of magnitude and squeezes the attribute ranges.

$$x' = log(1 + x) \tag{3.3.1}$$

Afterwards, a z-score normalization is applied to the data to obtain zero-mean, unit-variance distributions which has been shown to be optimal for gradient-based learning [146]. The aspect of data scaling is further discussed in details in the next chapter.

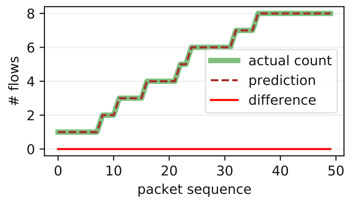
### 3.3.3 Evaluation Metrics

The model is trained to detect the start of each new flow and increment its output respectively. We use the MAE for measuring the absolute distance of the prediction to the actual value. We additionally use the Mean Absolute Percentage Error (MAPE) for a more intuitive interpretation of results. The MAPE represents a marginal error per sample in %. Given the true prediction $y_{i,j}$ of the the $j^{th}$ timestamp for the $i^{th}$ sequence, and $\widehat{y}_{i,j}$ being the respective LSTM output, $M$ the number of sequences in a test set and $N$ the length of the input sequence, the MAPE is computed as follows:

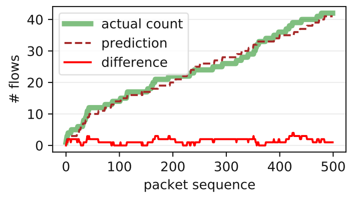
$$MAPE = \frac{1}{M}\sum_{i=0}^{M-1}\left|\frac{\widehat{y}_{i,N} - y_{i,N}}{y_{i,N}}\right| \times 100 \quad [\%] \tag{3.3.2}$$

The MAPE allows to interpret the error as number of flows in excess or absence. Moreover, it weights results and makes them comparable when dealing with sequences of different number of flows on average. Last but not least, we define an accuracy score that reflects the performance after observing a large number of sequences (an entire test trace in our case). This score represents a straightforward comparison between the final predicted flow count and the real count. It is defined as:

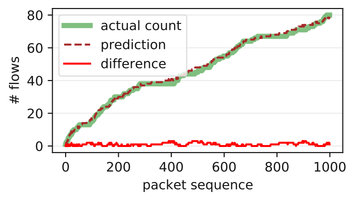
$$Accuracy = \left(1 - \left|\frac{\sum_{i=0}^{M-1} y_{i,N} - \sum_{i=0}^{M-1} \widehat{y}_{i,N}}{\sum_{i=0}^{M-1} y_{i,N}}\right|\right) \times 100 \quad [\%] \tag{3.3.3}$$



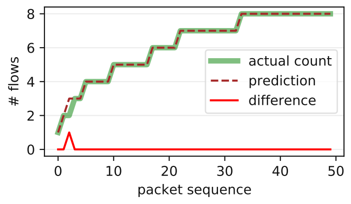
**(a)** Seq. length: 50 packets.



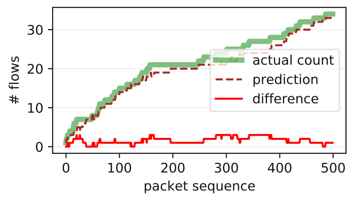
**(b)** Seq. length: 500 packets.



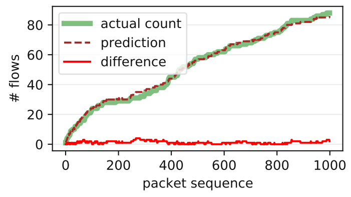
**(c)** Seq. length: 1000 packets.



**(d)** Seq. length: 50 packets.



**(e)** Seq. length: 500 packets.



**(f)** Seq. length: 1000 packets.

**Figure 3.4:** Random samples chosen to visualize the estimated flow count. The prediction closely follows the ground truth showing minimal error. Note that the number of flows in a sequence is an accumulative measure (i.e., it can only increase).

## 3.4 Results and Discussion

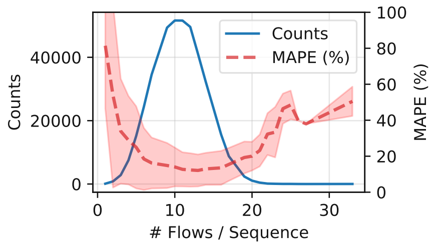
We train multiple vanilla LSTM networks and select the one with the minimum MAE loss for each sequence length (50, 100, 500 and 1000) after a given number of epochs. To obtain interpretable results we de-normalize predictions and data for the evaluation.

### 3.4.1 Overall Performance

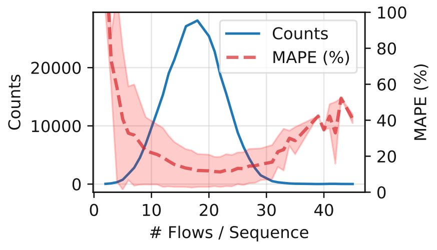
Figure 3.4 shows six random sequences picked from the test set of length: 50, 500 and 1000 packets. Curves depict the actual increase in the number of flows in a window as well as the model prediction after each packet. The figure illustrates how the predictions accurately match the ground truth with small to no error margin (represented by the brown line).

In Figure 3.5, we show the average MAPE value over all samples in the test set (Test 1) for a given "number of flows per sequence". For example, in Figure 3.5a, we use 50 packet sequences and find that there are approximately 50k sequences that contain 10 concurrent flows. We analyze this statistic for various sequence lengths and notice a smaller MAPE at high flow counts; this indicates that the model is trained better for such instances because we have more training samples (aka sequences). Thus, the minimum MAPE is obtained for sequences with the most frequent "number of flows per sequence".

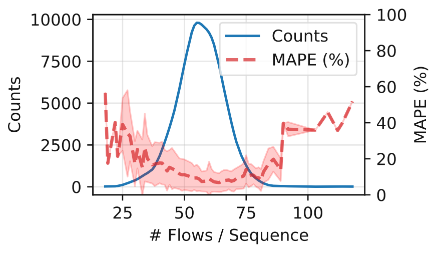
We show various error measures discussed in Section 3.3.3 in Table 3.3. The best MAPE score (9.06%) is obtained on the validation set for sequences of 1000 packets, which means that a real sequence of 1000 packet has an average number of 90.47 flows and the output



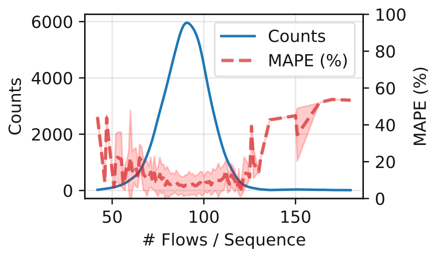
**(a)** 50 packets.



**(b)** 100 packets.



**(c)** 500 packets.



**(d)** 1000 packets.

**Figure 3.5:** The average MAPE for each "number of flows per sequence" value for various sequence lengths. The count represent the number of sequences from the test set with a given "number of flows per sequence".

of the LSTM is on average off by approximately 9.19 flows (MAPE × AFS.) therefore predicting a value between 81.28 and 99.66.

Furthermore, the longer the sequence length is, the smaller the MAPE and the better the accuracy are. Such behavior can be justified as a consequence of the *reversion to mediocrity* effect; in other words, the more data we have (past knowledge), the more representative learned patterns and statistical estimations become. Besides that, missing a few flows in a long sequence has a smaller effect than missing the same number of flows in a shorter sequence. However, the error increases and the model performance drop when guessing flows with data from a different time frame (Test 1) or a different year (Test 2).

Results above indicate that longer sequences are better suited to count flows. But note that using extremely long sequences can cause error accumulation due to the fact that previous (poor) cell states are continuously used as "inputs" in the LSTM. Moreover, the LSTM model requires a fixed length to delimit a meaningful context scope and can also suffer from exploding/vanishing gradient problems [185]. The study of optimal observation windows is one of the challenges to be tackled in subsequent work.

### 3.4.2 Sensitivity Analysis

In the design of any ML-based predictor for security applications, robustness against adversarial attacks becomes highly relevant. Attackers may be interested in manipulating

| Dataset | Sequence | AFS* | MAPE** ↓ | MAE ↓ | Acc. ↑ |
|---|---|---|---|---|---|
| **Validation** | 50 | 10.58 | ±15.95% | 2.42 | 97.05% |
| | 100 | 17.54 | ±14.73% | 5.57 | 97.62% |
| | 500 | 56.46 | ±10.23% | 7.29 | 98.05% |
| | 1000 | 90.47 | **±09.06**% | 8.18 | 98.89% |
| **Test 1** | 50 | 22.54 | ±22.61% | 5.43 | 97.05% |
| | 100 | 41.04 | ±21.90% | 9.41 | 97.05% |
| | 500 | 164.05 | ±18.60% | 13.95 | 97.05% |
| | 1000 | 291.318 | **±17.23**% | 18.57 | 97.05% |
| **Test 2** | 50 | 26.14 | ±23.64% | 6.18 | 76.57% |
| | 100 | 48.01 | ±21.00% | 10.55 | 77.16% |
| | 500 | 195.30 | ±19.52% | 13.46 | 80.05% |
| | 1000 | 346.093 | **±19.23**% | 21.56 | 82.05% |

∗ AFS: Average # Flows per Sequence.

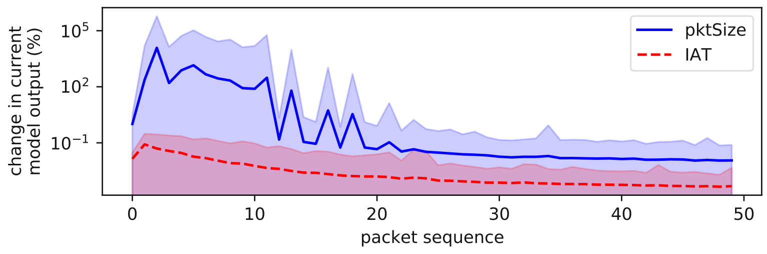
∗∗The MAPE represents the margin regardless of the sequence length or the number of flows in a sequence, e.g., if we consider a dataset with 100k flows and use 500-packet sequences (MAPE=9.06%), we get a margin of ± 9.06k flows i.e., an estimate between 90.94k and 109.06k flows.

**Table 3.3:** Performance evaluation over three datasets. The arrows indicate whether a high or low score is preferable.

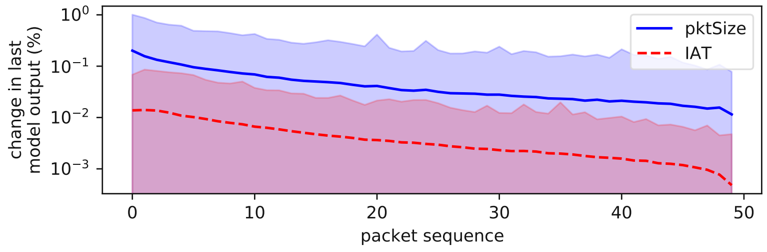
traffic to conceal their malicious activity or falsify statistics. In this section we test the stability of our proposal against variations in the relevant features. For this purpose, we carry out a sensitivity analysis inspired by [104].

Partial Dependence Plots (PDPs) [89] build global model explanations by fixing one feature and altering the rest over all possible combinations. In our case, we slightly change this method and build a perturbation model that shows the change in the model output when a single feature is manipulated over a defined range. We alter the packet size and the inter-arrival-time up four folds (note that an attacker could only increase those values because the delay can't be shorter than the network's actual delay). Because our model deals with sequences rather than a static input and predictions occur after each time step, we define two types of perturbation effects: local perturbation and global perturbation. Let $\mathbf{x}_{i,j}$ be the four-dimensional input of the the $i^{th}$ sequence at the $j^{th}$ time step and $y_{i,j}$ its respective ground truth, $N$ being the sequence length, $k$ and $p$ the current sequence and current time step indices respectively, we define:

- The Local Perturbation is the amount of change in the LSTM current output ($\Delta\widehat{y}_{k,p}$) caused by variations in the current input ($x_{k,p}$).
- The Global Perturbation is the amount of change in the LSTM last output ($\Delta\widehat{y}_{k,N}$) caused by variations in the current input ($x_{k,p}$).



**(a)** Perturbation effect on current prediction.



**(b)** Perturbation effect on last prediction.

**Figure 3.6:** Perturbation analysis of the LSTM output for sequences of length 50. At each sample, the packet size and inter-arrival times are manipulated up to 400%.

We show both perturbations ($\Delta y_{i,j}$ and $\Delta y_{i,N}$) for sequences of length $N = 50$ in Figure 3.6a and Figure 3.6b. Curves represent average values across the entire dataset (200k sequences), while shaded areas represent standard deviations. Both figures illustrate that the magnitude of the perturbation usually decreases toward the end of the sequence, which can be explained by the nature of the model used. LSTMs tend to retain knowledge from previous time steps, therefore new samples affect the prediction less if the previous knowledge is genuine. In short, the later in a sequence a manipulation occurs, the less effect it has. A second observation is that variations in packet lengths have a larger impact than variations in inter-arrival times. We hypothesize that this is caused by the stronger discriminant power of packet length to characterize network traffic [80, 153]. Finally, we observe that perturbations have a higher impact on the local outputs (Figure 3.6a) than on the final output (Figure 3.6b), which suggests that the final prediction at the end of the sequence is more robust against manipulations in general.

In any case, packet length is difficult to manipulate in encrypted network traffic if the attacker does not have access to end hosts because IP header checksums of the original packet must be recalculated, which requires decryption first.

## 3.5 Conclusion

In extreme cases, such as IPsec, nearly all conventional features used for analysis are encrypted, demanding the development of a new representations and a new generation of analysis methods. In this chapter we present a preliminary manual solution for counting encrypted flows in network tunnels. We then propose a ML-based approach to solve the same task but in a more complex setup, and show that our proposed representation combined with LSTMs can overcome the complexity limitations of manual approaches as they learn multiple patterns while only retaining relevant information from past data.

Our experiments over two backbone datasets with various sequence lengths reveal that the proposed solution performs satisfactorily with $\approx 9\%$ error margin despite encryption. The same model applied on data from a year later still obtains $\approx 19\%$ error margin, indicating that trained models are temporally transferable for certain applications with low error, thus requiring only long-term adjustment and fine tuning. Beyond flow counting, future work undertakes using the same representation and flow count estimation for solving more complex tasks such as clustering packets based on the flow they belong to.

CHAPTER 4

# Influence of Data Transformation Methods to Feature Spaces and Detection Performance

Thus far, we have proposed several data representations to improve input space construction, allowing for better separation of network traffic classes. Nevertheless, scaling the data to fit specific requirements imposed by the learning scheme used later is an essential aspect of a better construction of the input space along the way.

## 4.1 Introduction

In this chapter, we investigate the effect of multiple input space transformation techniques in an application where ML-based anomaly detection combined with a lightweight representation of network traffic statistics and meta-information is being used to detect various attacks. Generally, a trained classifier might achieve high detection accuracy even without much attention to this step or using standard techniques (e.g., Min-Max or Z-score scaling). However, in this chapter we confirm that the choice of the transformation technique indeed affects training speed, the model's interpretability, and in some cases, the convergence of the learning procedure.

Researchers often employ domain knowledge and visualization techniques to transform data into what they require; to exaggerate, reveal, or conceal a specific behavior, phenomenon, or pattern. Therefore, the borderline between the right and wrong way to do so is blurry. To address this, we further provide recommendations on how to scale the most commonly used statistical features extracted from network traffic with the most commonly public datasets for network attack detection. These recommendations do not serve as absolute rules because they are highly related to the features and dataset used.

In summary, our contributions are:

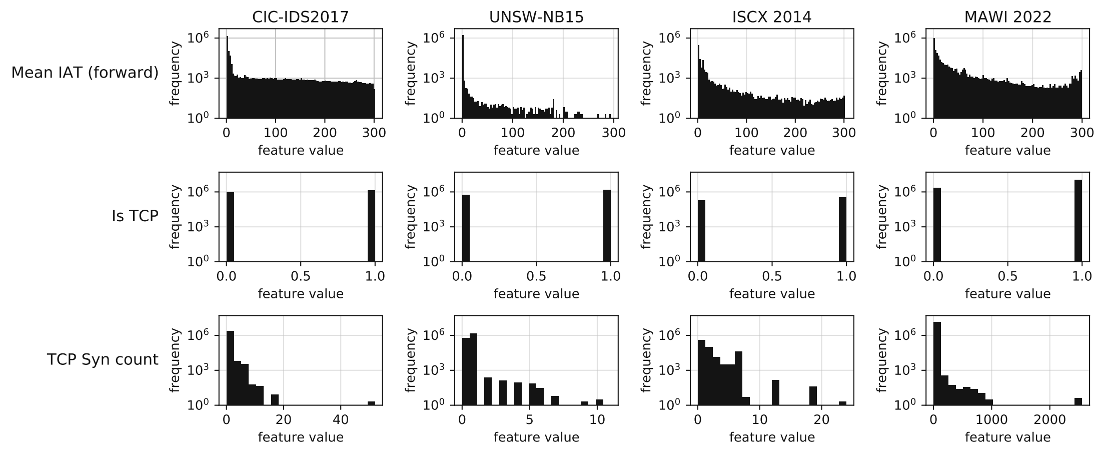


**Figure 4.1:** The three main distributions observed in network traffic across four datasets.

- We discuss a set of transformation techniques and analyze their effect on the input space and four main ML algorithm families.
- We present recommendations for dealing with feature scaling in the context of supervised learning for attack detection.

## 4.2 Typical Feature Distributions in Network Traffic

As mentioned before, statistical discriminators and meta-data extracted from packet headers and inter-packet times have been extensively used to identify traffic and attack patterns in network traffic. In [175], authors proposed a 249-dimensional set of stateful and stateless statistical features for representing traffic flows without the need for DPI. Several papers have recently repeatedly used a subset of these features to solve various problems related to NTA [85]. In this thesis, we further proposed several representations aggregating many traffic features together to aid in solving many of the open problems in this field. We have seen that most discriminators are derived by aggregating packets into network flows. Then meta-data is extracted from packet header fields and flow statistics, and further transformed by applying various statistical operations. We also discussed that packet payload is often excluded because it is often encrypted and thus inaccessible. We use the CAIA representation consisting of a subset of 22 features [246] in this chapter as a typical feature vector used in NTA. This feature set contains a reasonable number of features while performing well compared to other subsets [172]. Nonetheless, derived conclusions can also be applied to other representations such as the *multi-key* or I-Notice.

Network packets are generated based on well-defined protocols; therefore, network traffic typically exhibits the same pattern if averaged over a long period of time. In our investigation of several commonly used datasets we often observed similar distributions of the same features. We discuss in the following why is this the case. Figure 4.1 shows a set of three features extracted from four publicly available anomaly detection datasets, namely: ISCX 2014 [27], UNSW-NB15 [176], CIC-IDS2017 [220] and MAWI 2022[1] [164]. Although analyzed datasets are captured at networks with different configurations, bandwidths, and mutually exclusive timelines, the observed distributions are strikingly similar. Nonetheless, in the event of network errors, misconfigurations, or outages, the distributions may be affected and vary slightly but we haven't observed this behavior in the studied datasets. We classify the majority of studied features into three groups **based on the analyzed distributions** across four attack detection datasets (see Figure 4.1):

- *Size/Time-related features.* Features related to packet length or packet IAT are typically *Log-series* distributed, with plenty of small values and a long extended tail with fewer ones. Various factors induce this behavior: 1) Short IATs result from the desired low latency behavior when hosts/servers communicate with each other. 2) Small packets are the most common ones because they result from interactions with web-servers, applications and VoIP (mice vs. elephant flows [101]).
- *Binary features.* Typically, these features result from encoding categorical variables such as the protocol number, transport ports, or binary flags (e.g., TCP flags). They exhibit a binary distribution that is often imbalanced.
- *Count features.* Such features represent the occurrence of a given statistic in a flow and typically follow a Poisson distribution (with $\lambda = 1$) and a short range. One example of such features is the number of occurrences of a given TCP flag across all flow packets.

Surprisingly, none of the analyzed features follow a *Bell Curve* distribution, which is assumed by many learning algorithms, as shown in Figure 4.2.

In what follows, we treat each feature as a separate variable and do not consider the joint distribution of features. If the joint distribution of a subset of features is to be maintained, we recommend applying the same transformation to avoid breaking the mutual data structure. This point is further discussed in Section 4.3.3

## 4.3 Data Transformation Techniques

We first classify the most commonly used algorithms for attack detection based on their underlying learning scheme, then introduce typical transformation methods used to preprocess data, and finally present recommendations and examples on how each group of the aforementioned network traffic features can be processed based on the selected algorithm.

In our experiments different data transformation methods often did not affect the detection accuracy but had a more profound effect that is usually unobserved as discussed in what

[1]MAWI datasets are labeled based on a framework proposed in 2014 and has since been improved/updated. The traces used here, however, are from 2022.

follows. Most importantly, we discourage blind data manipulation irrespective of how well the model performs after training and we further provide examples of how data transformation can affect various aspects if done wrong.

### 4.3.1 Underlying Learning Scheme

We adopt the classification of ML algorithms into four major families according to [131]. This classification is based on the underlying learning scheme used to learn the objective function or to make a decision and, consequently, a prediction. Figure 4.2 depicts the four families discussed further below. Algorithms in green are typically affected by scales and require all features to be on the same scale. Algorithms in red or gray do not impose any conditions. However, several gray algorithms assume certain distributions at the input but are unaffected otherwise. SVM typically belongs to the regression-based family. However, it acts differently on the input space by computing distances among features; therefore, we classify it as a distance-based algorithm.

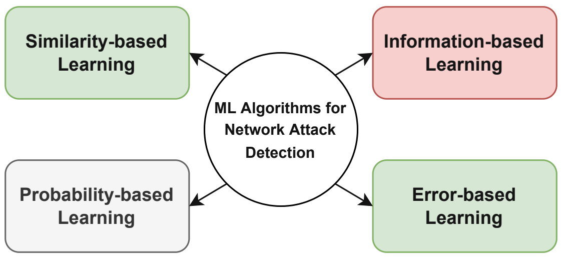

Similarity-based Learning
Information-based Learning
ML Algorithms for Network Attack Detection
Probability-based Learning
Error-based Learning

**Figure 4.2:** Algorithm families commonly used for anomaly detection.

**Similarity-based learning** The scales of input data have the most significant impact on algorithms that use distances to measure similarity or compute decision boundaries. This is because similarity metrics typically do not consider different dimensions separately but rather aggregate them into a single value, i.e., distance, so a dominant dimension can increase the distance significantly. This problem can be solved with some algorithms by using a custom distance metric, e.g., distances like the *Mahalanobis* distance incorporate a scaling factor into the computation and is thus unaffected by the scale. Other distance metrics, however, such as the widely used *Eucledian* distance, are severely impacted. k-NN, SVM with linear kernels, k-Means, and numerous other clustering algorithms fall into this category. In this case, it is preferable, if not required, that all input features have comparable scales; unless a given sample is intentionally secluded.

**Error-based learning** Models that use logistic units or gradient optimization based on an error quantity for the learning process are also influenced by the distribution (shape and values) of the data at the input space. Such algorithms require features in the same range, and a typical distribution for a variety of reasons [122], namely:

- When using weight penalization in a logistic regression solution, the scale becomes important as all weights must be relatively in the same range. Thus inputs proportionally affect the final output sharply [196]. Bringing all inputs down to the same scale is, therefore, essential.

- The initial weights in Multilayer Perceptron (MLP) networks are typically sampled from a normal distribution ($\mu$=0 and $\sigma$=1), requiring a similar input data distribution. Otherwise, several weights may explode or vanish quickly, making the learning process unstable and the gradient optimization oscillating [96] (Section 8.4).

- MLP networks learn the concept of numbers, which means larger values indicate higher importance. As a result, the interpretability of a model's predictions becomes improper and biased toward features with larger absolute values.

Consequently, in this case, it is recommended to transform the data to mimic a normal distribution and preferably standardize it so that it has a zero mean and a unit standard deviation [122]. In Section 4.3.2 we show selected techniques used to achieve this. Simple linear regression is excluded here because the input coefficients can already learn the scales since they do not incorporate regularization.

**Information-based learning** Tree-based models such as DTs, Gradient Boosting (GB), or ensembles such as RFs construct a set of rules inferred solely from a given feature value similar to if-statements; thereby, they do not use any distance or similarity measure between attributes. As a result, they are unaffected by the feature scale or its distribution.

**Probability-based learning** Probabilistic models, like rule-based ones, are typically unaffected by the scale of input data because they use approaches based uniquely on (conditional) probability estimation. As a result, NB, Markov Chain (MC), and many analogous algorithms don't need scaled data. Nonetheless, some variants assume a given distribution of the data to ease the computation of probabilities. Some special types of NB, such as the Gaussian Naive Bayes (GNB), which internally computes the mean and variance of each input feature, assume that they are normally distributed. Consequently, some work show that it is advantageous to transform the input features to conform to requirements about feature distributions [260].
Depending on the algorithm, recommendations should be respected regarding the scale and (less strictly) the data distribution in the input space. Nevertheless, this does not imply that a given model cannot handle the data otherwise; rather, transforming the input space accordingly aids in improving other aspects, such as faster training speed and stability, and yields better model interpretability.

### 4.3.2 Studied Methods

There are numerous space manipulation techniques typically referred to as scaling, normalization, standardizing, or transformation techniques. We divide them into three categories[2]:

- *Scaling.* It allows us to get all input data into a specific range and is sometimes called "normalizing". It works by adding or subtracting a constant from the input and multiplying or dividing it by a range [3].
- *Standardizing.* Unlike the previous one, this technique centers a given distribution at a specific coordinate by removing an offset and normalizing it with a scale unit.
- *Other Transformations.* These methods primarily affect data distributions and allow for an input space transformation. Some may be linear, while others may not be. They are typically used to correct distribution characteristics such as skewness, kurtosis, heteroscedasticity, or lack of normality.

Combining scaling and standardization boils down to the last operation performed, i.e., scaling followed by standardizing is equivalent to simply standardizing and vice versa. Other transformations, on the other hand, can be combined with either of the previous techniques. In practice, transformations are typically used to change a specific data characteristic before scaling or standardizing it. Although not impossible, combining a scaled subset of features and a standardized one is uncommon. It risks breaking the dependencies between features —at least in the case of network anomaly detection—.

We provide a few examples of each of the previously described methods in what follows.

#### Scaling

Examples of such techniques include Min-Max scaling, in which all values are scaled into the range [0, 1], and Max-Absolute scaling, in which all values are transformed into the range [-1, 1]. Other ranges can be obtained by adding the appropriate offset. These methods are suitable when the data already has a distribution that meets the algorithm's requirements, and instead, the goal is to have all dimensions on the same scale. The disadvantage of these techniques is that they are strongly affected by extreme values; thus, they are primarily used with data that is more or less uniformly distributed, such as pixel values in images.

#### Standardizing

**Z-Score standardizing** As many learning models assume normally distributed input data, this technique is commonly used among researchers to obtain a distribution with

[2]Depending on the domain, the terminology used in the literature might slightly differ.

[3]term "normalizing" is also frequently used to refer to the process of transforming a Gaussian distribution into a standard normal distribution ($\mu$=0 and $\sigma$=1), therefore we here use the term scaling.

zero mean and a unit standard deviation. Although not harmful, it is a common mistake to apply Z-Score to non-normal data. Since this technique does not affect the shape of a distribution but rather is considered as a translation of parameters, it is often not the optimal solution if the input data is not normally distributed. In such cases, a transformation can be used to fit the data into a Gaussian distribution before standardizing it, as discussed below.

**Robust standardizing** If a high portion of the data is normally distributed, but the rest includes extreme values, a more robust variant of the Z-Score standardization can be used. In this case, the computation uses the data's median instead of the mean and the inter-quartile range (the range between the first and third quartiles) instead of the standard deviation to overcome the effect of extreme values.

**Gelman-Hill (GH) standardizing** In [93] (Section 4.2), a slightly different standardization technique for binary data is proposed where the denominator consists of twice the standard deviation instead of one. Authors argue that the usage of this modified version aids in maintaining "coherence" to other input features when using binary inputs, allowing for a straightforward interpretation of the results, particularly in regression problems.

### Other Transformations

**Log transform** The logarithmic transform is a non-linear monotonic function that can be used to reveal information hidden in magnitude orders or to correct the skewness of left-skewed distributions. However, this transformation does not address other issues, such as the lack of normality or mitigating the impact of outliers, and it is not always appropriate. Recommendations how to use this transform can be found in [49].

**Quantile transform** A non-linear function that acts by dividing the data into quartiles and then using the cumulative distribution function to map the original distribution into any desired distribution theoretically. The quantile transform is commonly used to shape a given set of values into a Gaussian distribution and, henceforth, reduce the effect of extreme values. However, it occasionally performs poorly when dealing with unevenly distributed data (bulks of samples).

**Power transforms** Another type of monotonic functions that fits the data into a Gaussian distribution. These techniques aid in improving the symmetry of the distribution, particularly if it is skewed, as well as stabilizing the variance by addressing the distribution's *heteroscedasticity*. Many methods have been proposed, with the Box-Cox transform [37] being one of them —also used in this work—. However, this technique distorts the input space and shall be used cautiously.

**Feature clipping** It is a well-known technique for removing extreme values by simple maxing-out all values that are larger than a certain value. It, therefore, allows a more relaxed usage of subsequent algorithms that are affected by outliers at the price of information loss.

### 4.3.3 The Effect of Transformation for a Common NTA Example

Frequently, transformations are applied to transform the input space into a given shape or for the subsequent algorithm to pick a suitable data pattern. Here, we give the most appropriate recommendations to scale network traffic data to optimize it for various anomaly detection algorithms. A combination of a functional transformation followed by scaling or standardization is generally used. That is, based on the requirements/recommendations imposed by the algorithm, certain characteristics of the data are initially tuned before applying a scaling technique. Once a transformation is applied, we have three options: (1) scaling, (2) standardizing, or (3) simply nothing. We summarize the essential characteristics of each technique discussed in Section 4.3.2, as well as prerequisites and advantages in Table 4.1.

Consider a traffic representation (e.g., CAIA) with an MLP-based attack detector (e.g., Feed-forward Neural Network (FNN)). We apply previous recommendations in a supervised environment to detect network traffic anomalies. In Section 4.3.1 we explain how these models prefer normally distributed data for training; we, therefore, opt to transform each group of the features discussed in Section 4.2 as follows:

- Size/Time-related features. We apply a Quantile transform followed by a Z-score standardization to mimic a standardized normal distribution. The reason for choosing a Quantile transform is because it can both transform any given distribution into a normal one and simultaneously handle outliers, which we refer to as extreme values in this case (illustrated by the long tail of some distributions).
- Count features. We use a power transform followed by Z-Score standardization. This choice is because a normal distribution is suited, but the original distribution does not have many distinct values (a Quantile transform would hence fail to mimic a normal distribution in this case).
- Binary features. A simple GH standardization is used.

In the following, we compare the performance of this example with other typical data manipulation combinations.

## 4.4 Evaluation

For the evaluation of the example given above, we divide our experiments into three parts: (1) evaluation of the model's performance, (2) analysis of training convergence speed, and (3) checking the model's interpretability.

The evaluation steps are as follows:

**Table 4.1:** Techniques: pros, cons, and characteristics.

| | Scaling (Min-Max) | Z-Score Standardizing | Power Transform | Quantile Transform | Robust Standardizing | Log Transform | GH Standardizing |
|---|---|---|---|---|---|---|---|
| Preferably used when data is: | uniformly distributed | normally distributed | no preference | no preference | normally distributed with outliers | no preference | binary |
| Main benefit | reduces data range | centers and scales data | fixes heteroscedasticity | transforms distributions into an arbitrary shape | centers and scales data ignoring outliers | reveals percentage change in data | centers and scales binary data |
| Transforms distribution* | × | × | ✓ | ✓ | × | × | × |
| Limits data in a range | ✓ | × | × | × | × | × | × |
| Affected by E.V.** | ✓ | ✓ | × | × | × | × | × |
| Keeps E.V. proportions | ✓ | × | × | × | ✓ | × | × |

* Transforming the distribution into a Gaussian distribution (Bell curve).
** E.V.: Extreme Values.

- First, we extract flows in the CAIA format from datasets mentioned in Section 4.2.

- Then, we train a supervised classifier to detect network attacks based not only on a FNN but also an algorithm picked from each of the other three families discussed in Figure 4.2, namely: NB, DT and k-NN (with Euclidean distance).

- After that apply the following transformations: (a) Min-Max scaling, (b) Z-score standardizing, (c) Robust standardizing, (d) Quantile transform followed by Z-score standardizing, and (e) power transform followed by Z-score standardizing.

- Next, we compare the performance of all combinations, in addition to the analyzed example in Section 4.3.3, aka "enhanced transformation". To assess the model's classification performance, we split the data into training and testing sets and then employ the macro F1 score to measure the classification performance on the test sets. The F1-macro score reflects the model's actual capabilities without considering the data imbalance observed in the majority of IDS datasets. We average the final result over multiple runs to measure training time and obtain consistent results. Furthermore, we use the time needed to train a model with the original training set as a baseline with a time of $T$. Other techniques are either slower or faster than the baseline ($T$) by a given percentage factor. In the case of neural networks, the training time is measured as the time required to achieve the same accuracy obtained by the baseline in 32 training epochs.

- Following that we use SHAP [160] for the interpretability analysis. This allows us to measure the effect of scaling and whether untransformed features have a larger contribution to the prediction. We average the SHAP contribution scores for the MLP-based detector over a balanced set of random input samples from each attack family. These scores represent how each feature contributes to the prediction in the case of each transformation. We then compute the standard error over all input features thus, obtaining only a mean and a standard deviation for each transformation technique.

- We train a model for each scenario with fairness in the evaluation benchmark. This includes ensuring that the initial model's weights are similar, isolating OS threads for precise time measurements, and seeded operations whenever possible, e.g., when selecting a random set of samples to build the local SHAP explainer. All parameters, models, and data are available for reproducibility[4].

## 4.5 Results and Discussion

We evaluate the performance of each model on three levels: detection performance, training speed, and interpretability.

**Table 4.2:** Macro F1-score.

| Method | CICIDS-2017 | | | | UNSW-NB15 | | | |
|---|---|---|---|---|---|---|---|---|
| | MLP | NB | DT | kNN | MLP | NB | DT | kNN |
| Without | 0.306 | 0.175 | 0.677 | 0.262 | 0.171 | **0.091** | **0.579** | 0.205 |
| Min-Max | 0.629 | 0.290 | 0.674 | 0.291 | 0.343 | 0.015 | 0.578 | 0.218 |
| Power Transform + Z-Score | 0.559 | 0.242 | 0.675 | 0.533 | **0.365** | 0.024 | 0.575 | 0.417 |
| Quantile Transform + Z-Score | 0.627 | 0.293 | **0.681** | 0.488 | 0.352 | 0.078 | 0.575 | 0.352 |
| Robust Standardizing | 0.619 | 0.283 | 0.673 | 0.456 | 0.314 | 0.082 | **0.579** | 0.336 |
| Z-Score | 0.612 | 0.256 | 0.676 | **0.547** | 0.325 | 0.015 | 0.578 | 0.407 |
| Enhanced | **0.652** | **0.393** | 0.674 | 0.535 | 0.311 | 0.052 | 0.578 | **0.418** |

### 4.5.1 Detection Performance

Table 4.2 summarizes detection performance. It illustrates the F1 scores for seven different variants, two datasets, and four algorithms. Four observations stand out:

1. If no transformation is applied, the MLP-based classifier suffers greatly. The training loss function shows non converging quantities, and gradients begin to oscillate after a few epochs. This validates the assumption in Section 4.3.1 that FCNNs gradient optimization is highly affected by the distribution and scale of the data. Contrarily, it performs comparably well when any other technique discussed is used.

2. Whether scaling is applied or not, the DT-based classifier achieves the same score across all variants. This intuitively confirms the presumptions in Section 4.3.1.

3. When no transformation is applied, the NB-based classifier produces slightly lower results. It does, however, improve with the enhanced transformation. This is presumably due to the transformations applied to the data, which break the dependencies between features, eventually leading to independent features. This is then reflected in better performance because NB typically assumes independent features. It is worth noting that the correlation between input features reaches $0.\overline{9}$ in some cases and averages to 0.21 and 0.18, respectively, for both datasets (The average correlation is computed as proposed in [12]).

4. The performance of the k-NN-based classifier varies depending on the technique used. As expected, the classifier performs poorly when no transformation or Min-Max scaling is. This is because the relative distances are preserved in both cases; precisely, the computed distances are scaled in the latter. As a result, the algorithm fails against samples with dominant features (comparably larger values in one dimension.)

Previous observations apply to both datasets except that models trained on the UNSW-NB15 dataset achieve, on average lower scores.

[4]github.com/nta_preprocessing

**Table 4.3:** Training time in percentages. 50% means the training is completed in half of the time required by the baseline.

| Method | CICIDS-2017 | | | | UNSW-NB15 | | | |
|---|---|---|---|---|---|---|---|---|
| | MLP | NB | DT | kNN | MLP | NB | DT | kNN |
| Without | 100% | 100% | 100% | - | 100% | 100% | 100% | - |
| Min-Max | 67% | ≃ 100% | ≃ 100% | - | 70% | ≃ 100% | ≃ 100% | - |
| Power Transform + Z-Score | 59% | ≃ 100% | ≃ 100% | - | **55%** | ≃ 100% | ≃ 100% | - |
| Quantile Transform + Z-Score | 64% | ≃ 100% | ≃ 100% | - | **57%** | ≃ 100% | ≃ 100% | - |
| Robust Standardizing | 79% | ≃ 100% | ≃ 100% | - | 96% | ≃ 100% | ≃ 100% | - |
| Z-Score | 65% | ≃ 100% | ≃ 100% | - | 72% | ≃ 100% | ≃ 100% | - |
| Enhanced | **48%** | ≃ 108% | ≃ 100% | - | **58%** | ≃ 100% | ≃ 100% | - |

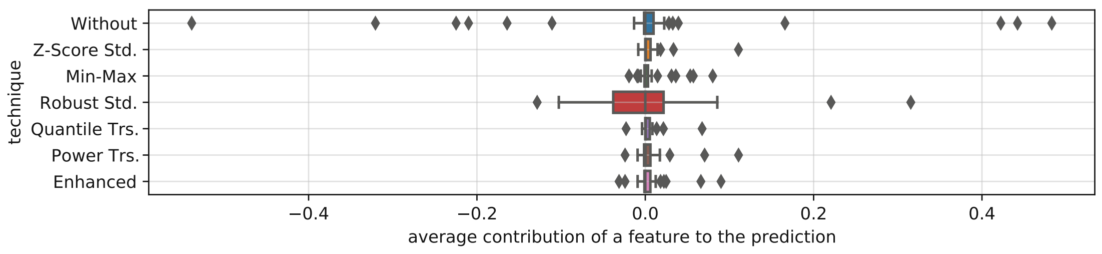


**Figure 4.3:** The effect of features on the prediction when applying each scaling technique.

### 4.5.2 Speed

We measure the training speed defined here as the time required to reach the baseline accuracy for the MLP model, the time required to learn the conditional probability model of the data for NB, and the time required to construct the tree for the DT model. k-NN belongs to the lazy learning algorithms, which do not require training but only store the data and the associated labels. Results are summarized in Table 4.3. To begin with, the MLPs-based classifier achieves the same accuracy as the baseline but faster when data is transformed. More specifically, the fastest variant occurs when data at the input space is normally (or close to normally) distributed. Other algorithms are not affected by the technique applied, show no effect on training speed, or their measured training time is nearly the same as the baseline. These same observations apply to the second dataset as well.

### 4.5.3 Interpretability

Figure 4.3 illustrates the mean and standard deviation of SHAP values for the trained MLP-based classifier. These values are averaged over all features and a random balanced selection of samples for each transformation. The x-axis reflects, on average, how a change in an input feature affects the prediction. Without any transformations, (some) features tend to affect the model's output sharply - in other words, the trained model becomes more sensitive to larger values which means a single feature risks flipping the prediction. This is unsuitable in practice because it makes the model prone to noise. When features are transformed into a similar range, the effect on predictions decreases, and we observe a balance in the contribution, yet features still have relatively different impacts. Robust standardization still has a significant standard deviation because, despite transforming the data, it theoretically ignores extreme values causing the model to encounter the same problem as with the untransformed samples discussed above. Ultimately, we observe how transformations can help stabilize interpretability results and reduce over-fitting on a subset of features. Therefore, applying a suitable transformation method may help build a robust model that is not strictly influenced by the feature magnitude.

## 4.6 Conclusion

In this chapter, we discuss an essential aspect in preparing data for ML algorithms, specifically for the case of network anomaly detection. We conduct several experiments to investigate the impact of multiple transformation techniques on various data characteristics. We show that transformations indirectly impact training speed, model stability, interpretability, and, to a lesser extent, classification performance. We also show that a transformation procedure derived from recommendations established by aggregating our conclusions and other work, performs relatively better for the network anomaly detection task with our datasets.

Future work in this area includes extending our conclusions to a broader range of applications, data representations, and problem types. Still, data preparation is a difficult task that must be done carefully and frequently necessitates a significant amount of external knowledge. We must note, however, that while our experiments gave interesting insights for the studied datasets, we cannot generalize conclusions in this regard; thus, as mentioned in the introduction, care must be taken when dealing with data transformation techniques.

# Summary

In Part II, I proposed three network traffic representations intended to solve three different anomaly detection tasks. In Chapter 1 I present the *multi-key* feature vector, a novel approach for aggregating network flows across multiple layers. In Chapter 2, I investigate the possibility of achieving similar performance without flow aggregation by proposing I-Notice. The list of proposed representations concludes with a minimal feature vector that aids in counting communication flows in VPN tunnels in Chapter 3. Finally, I present a set of experiments that investigate the impact of data scaling on the input space in Chapter 4. My proposals are not limited to anomaly detection but can also be applied to similar tasks such as protocol/application classification.

Throughout Part II, it became clear how vital feature engineering is for every subsequent operation, i.e., in most cases, the construction of the input space is a crucial aspect that determines the success of the proposed framework. Furthermore, combining the representation with a suitable ML algorithm is critical and must be carefully considered. The final chapter also confirms how data transformation improves performance in terms of training speed, model stability and interpretability results.

The following part focuses on improving the various components of an anomaly detection chain once data representation is agreed on. This includes sampling for faster training, data augmentation for better training, or faster neural networks for time-critical infrastructures.

# Part III

# Enhancing Detection and Classification Methods

# Overview

In the previous part of this thesis (Part II), I concentrated on developing feature spaces for network traffic analysis and anomaly detection. I investigated how proper input space construction can help improve classification and detection performance. In this part (Part III), I propose a few solutions and recommendations for improving the anomaly detection chain in various ways, regardless of the feature representation used. The proposals that follow address various components and, in some cases, extend existing solutions. More specifically, three problems are addressed here, namely: data sampling for faster analysis and training, data augmentation in an attempt to solve class imbalance and faster inference with DNNs. In the following, I provide a brief overview on each chapter:

**Proposal of a fast coreset extraction algorithm.** I introduce ODM, an algorithm for generating low-density models from large datasets. ODM assists in extracting data coresets, particularly from large network traffic captures, that can be used for stable and faster ML algorithm training. Coresets are a viable alternative to represent large datasets in that they preserve the main structure and relationships in the data, allowing for instance for faster testing of new algorithms without the need for the entire dataset. I present evaluation results demonstrating that ODM outperforms a set of other competitors in terms of accuracy-speed trade-off.

**Proposal of a generative neural network to tackle the problem of network traffic class imbalance.** As discussed in Part I - Chapter 3 - Section 3.6, network traffic data has some imperfections, one of which is data imbalance i.e., too many normal samples and a few malicious ones, which needs special handling during analysis and model training e.g., custom loss functions taking into account class weights. To address this, I propose *FlowGan*, a GAN-based framework that allows for configurable network traffic generation and augmentation. I compare various neural network architectures, as well as a well-established technique from the literature, and show that *FlowGan* performs better across various tests. As an extension to *FlowGan*, I propose *PacketGan*, a similar but more elaborated architecture that allows for generating packet sequences rather than flow samples. The latter can generate an infinite number of packets on the fly.

**Proposal of a custom architecture for neural networks that allows faster inference.** Because neural networks have been used extensively in this work; the final chapter focuses on improving one aspect of these networks: inference speed. I propose *EagerNet*, an adjustment to vanilla DNNs that allows for prompt predictions by halting the forward-pass once the prediction confidence is high enough. On average, this allows for faster inference and less resource usage, saving time and energy. The proposal works by considering a prediction loss at each layer of the neural network. During back-propagation, the optimization process considers all layers instead of the output layer only. The evaluation shows promising results, particularly in detecting unsophisticated network attacks that do not necessitate complex classifiers thus, only a few layers are required. I achieve comparable performance to standard FCNN if all layers are evaluated.

Further, I briefly summarize the set of datasets and algorithms used in each chapter in Table 0.1.

| Chapter | Datasets | Algorithms |
|---|---|---|
| 1 | MDCG, OTDT, MCC | ODM, SDO, KMC, BGM, GNG, CNN, kNN, k-Means, CDD |
| 2 | CICIDS2017 | GAN, RF, SMOTE, LSTM |
| 3 | CICIDS2017, UNSW-NB15 | FCNN |

**Table 0.1:** A summary of datasets and algorithms used in each chapter.

The proposals in this part contributes in improving different aspects with satisfactory results. However, many questions remain unanswered and must be addressed in the future. I discuss specifically this topic in Section 1.2).

CHAPTER 1

# Fast Extraction of Data Coresets

***Notice of adoption from previous publications - 6***
*Parts of this section's content have been published in the following paper:*
*Fares Meghdouri, Félix Iglesias Vázquez, and Tanja Zseby. Modeling data with observers.* Intelligent Data Analysis*, 26(3):785–803, 2022*

Data—particularly in network traffic analysis—is a key resource for AI applications since generalization and prediction require considerable volumes of knowledge to build reliable Ground Truth (GT) and subsequently predictive models. In principle, the quality of machine learning models often improves if more high-quality data is available for training.

## 1.1 Introduction

Managing, processing, and analyzing such ever-increasing volumes of data is a pressing challenge. Many algorithms either (1) cannot cope with massive raw data on-the-fly (freezing or collapsing) or (2) suffer from performance degradation due to noise, therefore requiring lightweight data summarization in between. For example, Zimek et al. show how sub-sampling in instance-based methods improves accuracy for outlier detection [263], introducing the counter-intuitive concept of "higher accuracy with less data". Henceforth, data summaries become key middle boxes for future ML, either to understand data, avoid degradation, or to reduce computational costs. A few models however are considered as an exception, e.g., with DL, large amounts of data are still needed because such algorithms use a brute force approach to learn patterns.

A way of summarizing data is by extracting a low density model from it, which is equivalent to what is commonly known as a coreset. A coreset is a subset $\mathcal{C}$ of data samples from an original set $\mathcal{X}$ that retains some characteristic properties for a given objective or fitting function $o(.)$ such that $o(\mathcal{C}) \approx o(\mathcal{X})$, as introduced by Agarwal et al. in [6].

To this end, we propose ODM in this chapter, a lightweight algorithm for extracting low density data models that scales properly and helps to simplify and understand huge volumes of both static and stream (network) data, creating optimized models for clustering, outlier analysis, and classification. ODM retrieves the idea of *data observers* presented in [233], which is a subset of data points used for approximating the original distribution, hence a coreset. Moreover, ODM admits some parameterization to adjust the granularity of the desired model and leverages the fact that observers do not necessarily have to be real data points.

## 1.2 Proposal: Modeling Data with Observers

This section describes the ODM algorithm explicitly, with a few visual examples, shows its complexity and elaborates on its mathematical formulation. The algorithm's framework is open-source and available online[1]. Table 1.1 collects notations and algorithm parameters used throughout this part.

**Table 1.1:** Notation.

| Parameter | Symbol | DV | Description |
|---|---|---|---|
| | $\mathcal{X}$ | – | Dataset |
| | $\mathcal{C}$ | – | Set of observers (coreset) |
| | $\mathcal{Y}$ | – | Random subset of $\mathcal{X}$ |
| | $\mathbf{x_i}$ | – | $i^{th}$ data point in $\mathcal{X}$ |
| | $\mathbf{c_j}$ | – | $j^{th}$ observer in $\mathcal{C}$ |
| | $\mathbf{y_k}$ | – | $k^{th}$ data point in $\mathcal{Y}$ |
| | $n$ | $\lvert\mathcal{X}\rvert$ | Cardinality of $\mathcal{X}$ |
| ✓ | $m$ | – | Cardinality of $\mathcal{C}$ |
| | $N$ | – | Space dimensionality (number of features) |
| ✓ | $R_o$ | – | Default radius of a new observer |
| | $R_k$ | – | Radius of a $k^{th}$ observer |
| | $P_j$ | – | Number of points assigned to the $j^{th}$ observer |
| ✓ | $f$ | 0.1 | Expansion coefficient |
| ✓ | $\rho$ | 0.025 | Sampling ratio for $R_o$ estimation |
| ✓ | $\beta$ | 1 | Correction factor for $R_o$ estimation |
| ✓ | $\alpha$ | 1 | Observer inertia coefficient |
| | $d(.)$ | – | Distance metric |
| | $O(.)$ | – | Time complexity |
| | $O_m$ | – | Time complexity of ODM with a M-Trees core |
| | $r$ | – | Size ratio (%) between $\mathcal{C}$ and $\mathcal{X}$ |
| | $\Delta u$ | – | Displacement of an observer after an update |

DV: Default value.

[1] github.com/CN-TU/pyodm

### 1.2.1 Observers

In [233], $C$ (the observers set) is initially formed by randomly sub-sampling $\mathcal{X}$. Authors define an observer as "*a data object placed within the data mass and ideally equidistant to other observers within the same cluster*". Therefore, applied as models, observers are intended to evaluate data points from the input space that are located in their vicinity. We take this intuitive idea as a basis and modify concepts by giving observers some new properties. Here, observers are not required to be actual data points. Additionally, we assign to each observer $\mathbf{c_j}$ a maximum observation radius $R_j$ and a population $P_j$, which accounts for the number of observed data points.

### 1.2.2 Algorithm

The goal of the ODM algorithm is to build a set of proper observers; aka an optimal coreset $C$ that maximizes evaluation metrics compared to the original set and retains some characteristic properties for a given objective function $o(.)$, as exposed in [6]. Therefore:

$$o(C) \approx o(\mathcal{X}) \tag{1.2.1}$$

with $\mathcal{X}$ being the original set.

ODM starts taking a random existing data point as the first observer. Later, it processes the remaining data points sequentially as follows. Given the $i^{th}$ data point $(\mathbf{x_i})$,

- if the distance to the closest observer $(\mathbf{c_j})$ is below the radius of the observer $(R_j)$, $\mathbf{c_j}$ properties are modified to better fit the data (i.e., the radius $R_j$ is shrunk and the location of $\mathbf{c_j}$ is slightly shifted toward $\mathbf{x_i}$);
- instead, if the distance is larger than the radius then $\mathbf{x_i}$, it becomes a new observer.

Some parameters to comment on are:

- $R_o$ (optional) is the default radius for any new observer. If not externally provided as a hyperparameter, $R_o$ is internally estimated (see Equation (1.2.9)). Large values of $R_o$ can cause superficial representations (very few observers), whereas very small values of $R_o$ reduce the compression rate and might result in more observers than necessary.
- $f$ is an expansion factor that defines the growth rate of $R$. Large values cause an unstable coreset construction because one observer can cover more space than needed, whereas low values causes the need for more observers to cover all the data.
- $\rho$ (required if $R_o$ is not given) is a sampling ratio used to estimate $R_o$ when this is not externally set. Higher values yield a better estimation but introduce a larger delay.

- $m = |\mathcal{C}|$ (optional) is the number of observers (size of the coreset). If the size of the coreset is not externally imposed with this parameter, ODM internally optimizes it by itself.

- $\alpha$ (optional) is a inertia coefficient that determines how observers update their locations. It enhances the configuration flexibility and set to 1 by default.

- $\beta$ (optional) is a correction parameter for estimating $R_o$. Also very robust, the default configuration obtained excellent performances when tested with a wide assortment of cases.

Algorithm 1.1 shows the core routine of ODM, its mathematical formulation is discussed in Section 1.2.5. The process depicted in Algorithm 1.1, explained in a more intuitive way, goes though the following steps:

1. Shuffle the dataset (to avoid bias).

2. If $R_o$ is not adjusted externally, a random subset of the dataset ($\mathcal{Y}$) is constructed with $|\mathcal{Y}| = \lceil n\rho \rceil$ elements. Then, $R_o$ is estimated as the average distance between all samples from $\mathcal{Y}$ and their corresponding centroid adjusted by a correction parameter $\beta$.

3. The coreset is initialized with a first observer $\mathbf{c_0}$, which is a randomly chosen data point from $\mathcal{X}$. Hence, $R_0 = R_o$, and $P_0 = 1$.

4. Later, the remaining data points are processed sequentially. For a given $\mathbf{x_i}$, the algorithm calculates the distance between $\mathbf{x_i}$ and its closest observer $\mathbf{c_j}$. If the distance is smaller than the observer radius $R_j$, $\mathbf{c_j}$ is moved closer to $\mathbf{x_i}$, its population $P_j$ increases by one and its radius $R_j$ is decreased by a fraction $f$ of the distance $d(\mathbf{c_j}, \mathbf{x_i})$. If, instead, the distance is larger, $\mathbf{x_i}$ becomes a new observer and $\mathbf{c_j}$ radius $R_j$ is increased by a fraction $f$ of the distance $d(\mathbf{c_j}, \mathbf{x_i})$.

The central loop of Algorithm 1.1 can be parallelized to enable faster extraction. Each data-batch is processed individually and the resulting subcoresets are joined together afterwards. Moreover, the current implementation of ODM allows for shuffling data points and picking them at random. Shuffling is useful when the arrangement of data samples has to be ignored; otherwise, samples originally sorted might belong to the same cluster causing the observer's radius to converge and shrink quickly, which ultimately creates more observers than needed. To avoid this and to enable a better space reconstruction, data points should be picked at random. Finally, if the size of the coreset is externally set as a parameter ($m$), the algorithm keeps only the $m$-most populous observers for the final coreset (sorted by their $P$ value).

**Algorithm 1.1:** ODM

```
1  (Required) Input: $f$, $R_o$
2  (Optional) Input: $\rho$, $m$, $\alpha$, $\beta$
   Output: $C$
   Data: dataset $\mathcal{X}$
3  Shuffle $\mathcal{X}$
4  if $R_o$ is not specified then
5      Estimate $\widehat{R_o}$ from $\mathcal{Y}$, a fraction $\rho$ of $\mathcal{X}$
6      Set $R_o = \beta\widehat{R_o}$
7  end
8  Set a random sample $\mathbf{x_r}$ as $\mathbf{c_0}$
9  Set $P_0$ to 1
10 Set $R_0$ to $R_o$
11 Append $\mathbf{c_0}$ to $C$
12 for $i$ in $|\mathcal{X}| - 1$ do
13     Get $r$ the index of the nearest observer to $\mathbf{x_i}$
14     if $d(\mathbf{c_r}, \mathbf{x_i}) \leqslant R_n$ then
15         Set $\Delta u$ to $\frac{\mathbf{x_i} - \mathbf{c_r}}{P_r + 1}$
16         Shift $\mathbf{c_r}$'s coordinates by $\alpha \cdot \Delta u$
17         Increment $P_r$ by 1
18         Subtract a fraction $f \cdot d(\mathbf{c_r}, \mathbf{x_i})$ from $R_r$
19     end
20     else
21         Add a fraction $f \cdot d(\mathbf{c_r}, \mathbf{x_i})$ to $R_r$
22         Create a new observer $\mathbf{c_{new}} = \mathbf{x_i}$
23         Set $P_{new}$ to 1
24         Set $R_{new}$ to $R_o$
25         Append $\mathbf{c_{new}}$ to $C$
26     end
27 end
28 if $m$ is specified then
29     Sort $C$ in descending order based on $P$ values
30     return first $m$ elements of $C$
31 end
32 return $C$
```

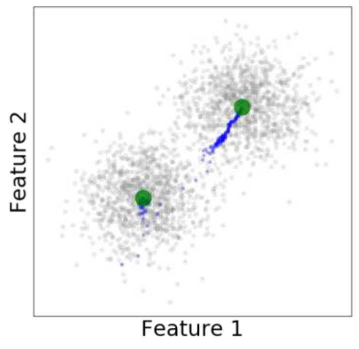


**Figure 1.1:** Two observers (in green) modeling two clusters. Gradual observer locations are shown in blue.

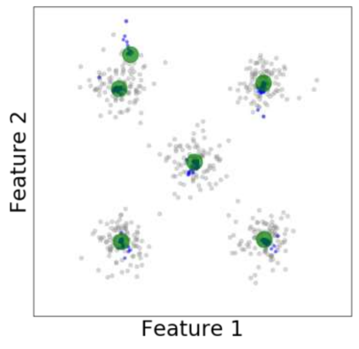


**Figure 1.2:** Six observers (green) modeling a five clusters. Gradual observers locations are shown in blue.

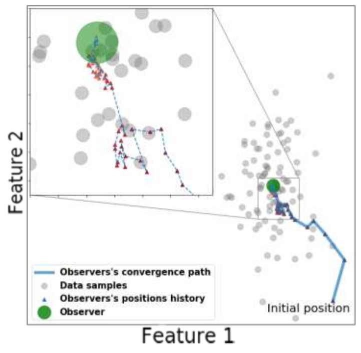


**Figure 1.3:** Convergence path of an observer to the center of a sample-cloud.

### 1.2.3 Examples

The operation of ODM can be better understood with a few simple examples. Figure 1.1 and Figure 1.2 represent two datasets [87, 199] with two and five clusters. For these two cases ODM has been applied to obtain coresets with two and six points respectively. Data points are shown in gray, the final positions of observers in green, and the displacement of an observer is tracked with blue dots. The figures illustrate how observers end up in approximately the respective cluster centers. If the number of observers is set to a higher number, the observers placement is expected to be distributed based on data density variations. We show a zoomed-in observer displacement in Figure 1.3 with a different dataset [88]. Here we can see that the observer location converges and its displacement gets progressively smaller.

The previous examples use very few observers in order to show the natural tendency of observers to be attracted by density centers. In practical modeling, higher granularity is commonly required, especially in cases with complex data structures. Figure 1.4 shows three datasets [128, 65, 232] for which ODM has been adjusted with low, medium, and high number of observers. As discussed above, denser coresets can be created by simply adjusting ODM parameters. Obviously, the size of a coreset cannot be larger than the size of the modeled dataset.

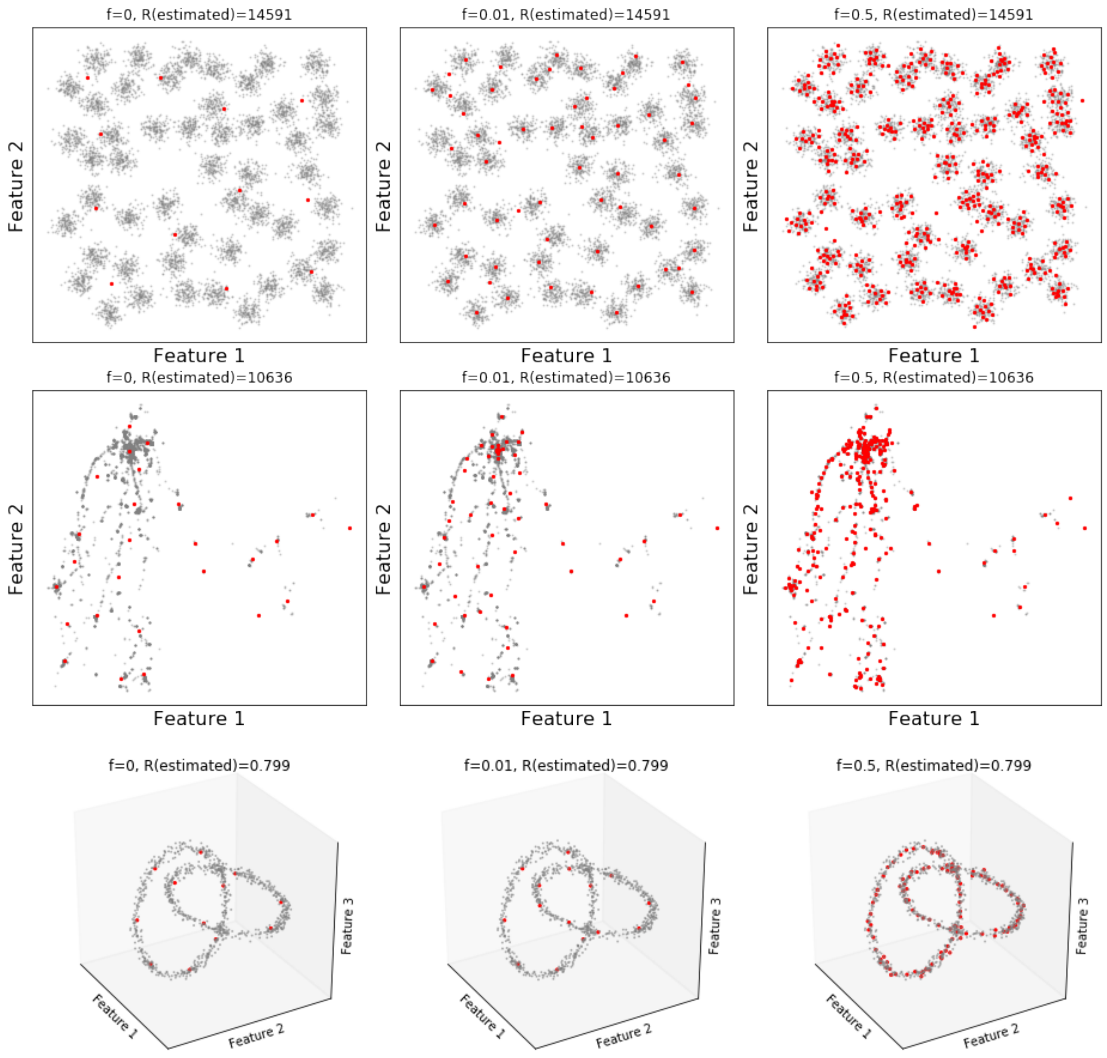


**Figure 1.4:** ODM coresets extracted from three different datasets. Each column represents a different parameterization resulting in a different number of observers. Data points are shown in gray and observers in red.

### 1.2.4 Complexity

The computational demand of ODM is mainly concentrated on the search of closest observers. Different algorithms can be used for this task; for instance kd-Trees [29] and M-Trees [55]. We opt for M-Trees in our implementation since they are robust, fast, and allow changes in the search tree without need for its reconstruction. The tree is built once at the beginning then grows and keeps track of the observers. Based on Algorithm 1.1, each new data point generates: one k-NN query, one insert (either a new observer or an (updated) existing one), and optionally one removal (remove an old observer if it has been updated and re-inserted). In the (improbable) average worst case scenario, which happens if the number of observers increases with each iteration and approaches the number of data points, the complexity $O_m$ with a M-Tree core can be estimated as follows: **number of data points** × **( k-NN query + removal + insertion )**, which gives:

$$\begin{aligned} O_m &= O(n) \cdot [O(logn) + O(logn) + O(logn)] \\ &= \mathbf{O(n\ log\ n)} \end{aligned} \tag{1.2.2}$$

$n$ being the number of processed data points (aka the cardinality of the dataset, $|\mathcal{X}|$),

### 1.2.5 Problem Formulation

In this section we address the mathematical formalism underlying ODM. Initially, assuming that at least one observer exists in a coreset $\mathcal{C}$, let $\mathbf{x_i} \in \mathbb{R}^N$ be the $i^{th}$ observation in an $N$ dimensional dataset $\mathcal{X}$ and let $\mathbf{c_j} \in \mathbb{R}^N$ be the closest observer to $\mathbf{x_i}$:

$$\mathbf{x_i} = \begin{bmatrix} x_{i,0} \\ x_{i,1} \\ \vdots \\ x_{i,N} \end{bmatrix} \qquad \mathbf{c_j} = \begin{bmatrix} c_{j,0} \\ c_{j,1} \\ \vdots \\ c_{j,N} \end{bmatrix} \tag{1.2.3}$$

If the distance between $\mathbf{c_j}$ and $\mathbf{x_i}$ is smaller than $R_j$:

$$d(\mathbf{c_j}, \mathbf{x_i}) \leq R_j \tag{1.2.4}$$

The coordinates of $\mathbf{c_j}$ are updated such that:

$$\mathbf{c_j} \leftarrow \mathbf{c_j} + \alpha \Delta \mathbf{u}_j \tag{1.2.5}$$

Where $\alpha$ is the observer's inertia coefficient that controls the displacement (set to 1 by default) and $\Delta\mathbf{u}_j$ is the displacement defined as:

$$\Delta \mathbf{u}_j = \frac{\mathbf{x_i} - \mathbf{c_j}}{P_j + 1} \tag{1.2.6}$$

$P_j$ is the number of data points associated to the observer $\mathbf{c_j}$ (i.e., observations/population) and $\mathbf{x_i} - \mathbf{c_j}$ is a distance vector. An observer $\mathbf{c_a}$ with high population is called *lazy* observer since the displacement when observing a new data point tends to vanish, i.e.,:

$$\lim_{P_a \to \infty} \Delta \mathbf{u}_a = \lim_{P_a \to \infty} \frac{\mathbf{x} - \mathbf{c_a}}{P_a + 1} = \mathbf{0} \tag{1.2.7}$$

Equation (1.2.7) shows that after many iterations, more observers become lazy and their displacement tends to vanish. Moreover, Equation (1.2.7) always holds since the distance vector $\mathbf{x} - \mathbf{c_a}$ cannot be arbitrarily large (in comparison to $P_a + 1$) nor exceeds the radius of the observer ( Equation (1.2.4)). The additional +1 in the denominator is utilized to make sure that initially an observer with population $P = 1$ moves half the distance toward the data sample observed.

The radius of the closest observer is at the same time updated regardless of the distance check performed in Equation (1.2.4):

$$R_j \leftarrow \max\left(R_j \pm f \times d(\mathbf{c_j}, \mathbf{x_i}), 0\right) \tag{1.2.8}$$

Depending on the distance check, the radius $R_j$ of an observer $\mathbf{c_j}$ is increased or decreased by a fraction $f$ of the distance between the current sample ($\mathbf{x_i}$) and the closest observer ($\mathbf{c_j}$) itself. Conversely the $max(., 0)$ operator ensures that the radius is always positive.

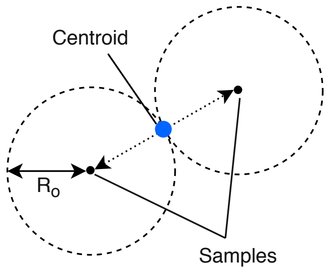


**Figure 1.5:** By applying Equation (1.2.9), the initial radius $R_o$ is estimated to be the radius of circles centered at each sample and optionally corrected using $\beta$ (=1 in this case).

If not specified as an external parameter, the initial radius of new observers is estimated as follows:

$$R_o = \beta\left(\frac{1}{|\mathcal{Y}|}\sum_{j}^{|\mathcal{Y}|} d(\mathbf{y_j}, \bar{\mathbf{y}})\right) \tag{1.2.9}$$

Whereby $|\mathcal{Y}| = \lceil n\rho \rceil$. Equation (1.2.9) computes the average distance to the centroid of samples in a set $\mathcal{Y}$ sampled from $\mathcal{X}$. $\beta$ is a tunable correction parameter (set to 1 by default). Figure 1.5 shows how $R_o$ is estimated in the case of a two-points subset.

ODM accepts different metrics (d(.)) for distance calculation. For instance, Aggrawal et al. [7] recommend using the Manhattan distances instead of Euclidean distances for high-dimensional spaces.

## 1.3 Evaluation

In this section we describe experiments conducted to evaluate ODM. They are tasks related to anomaly detection, clustering, and supervised classification. In the experiments, ODM is compared with alternative coreset extraction algorithms (described bellow), namely:

- **Baseline**. No coreset, the whole dataset is used instead.
- **RS**. A coreset obtained by uniform Random Sampling.
- **SDO**. Sparse Data Observers [233].
- **KMC**. k-Means Coreset [20].
- **BGM**. Bayesian Gaussian Mixture [158].
- **GNG**. Growing Neural Gas [90].
- **CNN**. Condensed Nearest Neighbours [102, 15] (only for supervised classification).

### 1.3.1 Coreset Extraction Methods

In what follows, we briefly go over some methods for extracting coresets. Some of these are used in this work.

#### Statistics and Error Optimization

Among the most used, Bayesian coresets [113, 43, 44, 42] sample data with probabilistic models to create subsets that mimic the whole data in terms of approximating the posterior inference. The problem can be differently tackled with Gaussian Mixture Models [18, 158]; here the shape of the original data is reconstructed in a coreset by using a mixture of Gaussian functions whose parameters are adjusted accordingly. Feldman et al. [83] propose an alternative method that imposes constraints driven by PCA and k-Means (k-Means) clustering. This method projects data into a low dimensional space, strategically selects some data points, and maps them back into the original space. The process ensures that both the whole dataset and the coreset yield similar principal components and centroids. Mirzasoleiman et al. [174] develop a technique for selecting a subset from a larger set with gradient-based learning algorithms. This method minimizes the error of an objective function in such a way that training with the entire dataset and the subset obtain the same error. Authors propose a greedy search algorithm to improve speed and avoid trying multiple combinations. The approaches introduced above build coresets very differently from each other, all of them also having a different theoretical basis from that used in ODM.

### Distance-based Methods

To generate coresets some methods combine smart data point sampling with measurements of point-to-point distances. For instance, Condensed Nearest Neighbour (CNN) [102, 15] was originally proposed as a solution for supervised training with a small number of representative samples. The algorithm is based on the 1-Nearest Neighbor label to retain only necessary samples and eliminate redundancy. The k-Means algorithm can also be used to find coresets [20] by extracting the shape of the input with a set of centroids that reflect major clusters in data. Finally, Iglesias et al. [233] mint the concept of “observers” to create a model of the data and use it later for detecting anomalies with the Sparse Data Observer (SDO) algorithm. Authors define observers as a set of sampled data points in which potential outliers have been removed according to a distance-based metric. These observers are later used as anchors to evaluate data points based on proximity calculations. ODM inherits the idea of “observers” from this work, but extends it by making them dynamically adaptable and controlling their generation and location. As a result, the set of observers in ODM is more density isotropic and homogeneous and less random.

### Neural Networks

Finally, techniques for extracting coresets with Neural Networks deserve a separate mention due to their peculiarities. Developed by Kohonen et al. [140], Self-Organizing Maps (SOMs) are strong alternatives for disclosing connections within multi-dimensional numerical data. They project complex input spaces into two-dimensional neural grids while trying to keep topological structures that are inherent to the original data. Based on SOMs, Martinez et al. [163] propose Neural Gas and, later, Fritzke et al. present Growing Neural Gas (GNG) [90]. These algorithms learn typologies by means of iterative fitting (or “growing”) processes. Intuitively, neurons behave like gas particles that extend and expand into the data, whose multidimensional shapes are seen as containers. This expansion of data points to occupy the entire space is analogous to the concept of *low density models* in ODM and play a similar role to the observers.

## 1.3.2 Datasets

Instead of using network traffic datasets that might contains only some specific data patterns, we decided to use a wide variety of public datasets that are commonly applied for algorithm testing in the literature to validate the utility of ODM. A prerequisite when selecting datasets was that they should contain a considerable number of samples in order to make the application of coresets meaningful. The datasets used are:

- **MDCG-datasets**. 15 datasets created with MDCGen [120], a tool for the highly-customized generation of multi-dimensional, multi-cluster datasets for algorithm testing. Parameters were selected to obtain scenarios with different number of dimensions, clusters, sample sizes, cluster overlap, and noise.

- **OTDT-datasets**. A set of six popular datasets for outlier detection collected from the literature [94, 3, 33, 81, 132, 253]. They are imbalanced datasets with two possible classes: normal and anomalies (or *inliers* and *outliers*). The positive class corresponds to anomalies, which account for less than 10% of the total dataset size.
- **MCC-datasets**. A set of five modern and balanced classification dataset with binary and multi-class targets (up to 10 classes) are used for the classification task [68, 126, 130, 46, 60].
- **Toy-datasets**. Some real and synthetic datasets (up to 3 dimensions) selected from the literature to illustrate characteristics of the algorithm under study [128, 65, 232, 87, 199, 88].

Table 1.2 shows characteristics (max and min values) of each dataset family. The noise level is calculated as the percentage of outliers. Whenever a dataset shows missing values, the corresponding sample is removed from the analysis. "na" stands for cases in which the GT was not given by dataset publishers.

**Table 1.2:** Datasets characteristics.

| Datasets | Samples | Dimensions | Classes | Ouliers | Noise (%) | Missing Values |
|---|---|---|---|---|---|---|
| MDCG | 1050-5250 | 2-20 | 2-20 | 50-300 | 1.96-9.09 | None |
| OTDT | 214-286048 | 3-27 | 2 | 9-3511 | 0.03-7.41 | Dropped |
| MCC | 4839-20867 | 5-11 | 2-11 | na | na | Dropped |
| Toy | 158-13467 | 2-3 | 1-51 | na | na | None |

### 1.3.3 Anomaly Detection

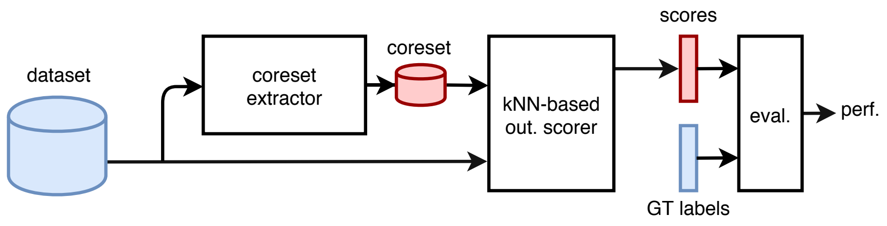


**Figure 1.6:** Scheme of the anomaly detection experiments.

Coresets can help anomaly detection to alleviate the computational cost of instance-based methods and avoid degradation [263]. The traditional way of measuring outlierness is by considering the whole dataset (or all previous data points) when evaluating each single data point. Instead, with coresets each single data point is contrasted only with the coreset to decide if it is considered as anomaly, therefore considerably reducing the computational burden. The experiment methodology is shown in Figure 1.6, its elements are described as follows:

- *Datasets.* We use OTDT-datasets (Section 1.3.2).
- *Coreset extractors.* Competitor algorithms were tuned to create coresets with different $r$ values (0.5%, 1%, 5% and 10%), where $r$ is the size of the coreset relative to the dataset size, i.e.,

$$r = \frac{|C|}{|X|} \times 100 = \frac{m}{n} \times 100 \qquad (1.3.1)$$

  Since anomaly detection is unsupervised learning, additional parameters of each algorithm are tuned with the default values suggested in the respective references.
- *Outlierness scorer.* We use the k-NN-based outlier detection algorithm proposed by Ramaswamy et al. [192] to obtain the internal outlierness scores. The number of neighbors is set to five (*k=5*).
- *Evaluator.* We evaluate performance by computing the P@n, average precision, ROC-AUC and MaxF1 scores using the GT labels as suggested by [45]. We show adjusted metrics to allow fair comparisons regardless of the number of outliers. We additionally measure coreset extraction and k-NN-based outlierness scorer (empirical) average runtimes.

The setup shown in Figure 1.6 covers 144 experimental runs (6 algorithms × 6 datasets × 4 $r$ values) in addition to the baseline variant. Results are summarized in Table 1.3 and discussed in Section 1.4.1.

### 1.3.4 Clustering

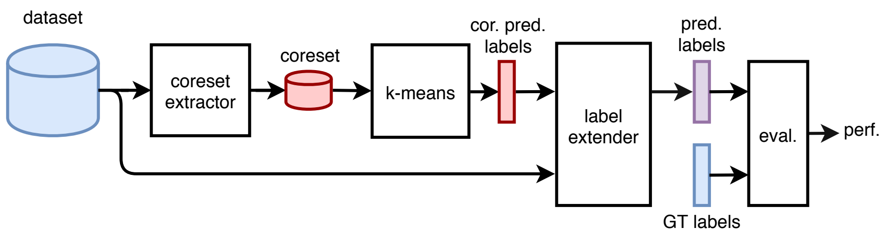


**Figure 1.7:** Scheme of the clustering experiments.

Similarly to the anomaly detection case, in order to reduce memory and time requirements, label assignment in clustering can be performed by clustering a coreset and extending labels to the whole dataset based on proximity calculations. We reproduce this scheme in our experiments and used k-Means as core algorithm. The methodology is shown in Figure 1.7 and its elements are:

- *Datasets.* We use MDCG-datasets (Section 1.3.2).
- *Coreset extractors.* Competitor algorithms were tuned with different $r$ values (0.5%, 1%, 5% and 10%). Again, additional algorithm parameters were tuned based on the default values suggested in source references.
- *k-means.* Clustering of the coreset data points is undertaken using k-Means++ [17]. The initial number of cluster $k$ is assumed as a known value prior to the analysis.
- *Label extender.* This block calculates coreset cluster centers (centroids) and assigns final labels to the whole dataset by extending the label of the closest center (k-NN, k=1).
- *Evaluator.* We evaluate performances with the external validity rand index [112] between predicted labels and GT labels and the internal validity silhouette coefficient [206]. We additionally measure coreset extraction, k-Means and label extender (empirical) average runtimes.

The setup shown in Figure 1.7 covers 360 experimental runs (6 algorithms × 15 datasets × 4 $r$ value) in addition to the baseline variant. Results are summarized in Table 1.4 and discussed in Section 1.4.2.

### 1.3.5 Supervised Classification

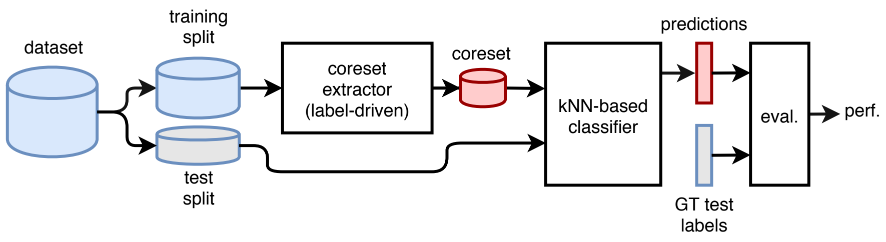


**Figure 1.8:** Scheme of the supervised classification experiments.

Coresets can also be used in supervised classification for data reduction, therefore suppressing noise, favoring generalization, reducing over-fitting and reducing training and/or evaluation complexity. Here coresets are intended to retain the essential set of training points for subsequent classification. Note that such data reduction is strongly recommended for instance-based (aka *lazy*) learners e.g., k-NN that deal with large datasets, in which the coreset would work as a model and make it closer to *eager* learning options. This approach would be consistent with the sampling recommendations given by Zimek et al. in [263] for the anomaly detection case. In our experiments we reproduce the setup with the following blocks (Figure 1.8):

- *Datasets.* We use all MCC-datasets (Section 1.3.2). Each dataset is divided into training (70%) and test (30%) splits with stratified sampling. Some datasets contain multiple classes, therefore we sometimes evaluate a multiclass classification problem.

- *Coreset extractors*. Competitor algorithms were tuned again with different $r$ values (0.5%, 1%, 5% and 10%). "label-driven" means that training data was divided into subsets based on GT labels and coresets were consequently extracted from each subset and later joined. This setup was used for all algorithms, except for CNN, which is already designed to properly deal with multiclass data. Also, vanilla CNN cannot be adjusted to create coresets with specific number of datapoints; therefore, random sampling was applied as a last step to equalize CNN with other algorithms and allow fair comparisons. Other algorithm parameters were adjusted by using Evolutionary search on training data.
- *k-NN-based classifier*. We train a vanilla k-NN-classifier to predict labels. The number of neighbors is set to five ($k$=*5*).
- *Evaluator*. We provide accuracy as well as micro and macro precision and recall scores. We additionally measure coreset extraction and k-NN-based classification (empirical) average runtimes.

The setup shown in Figure 1.8 covers 140 experimental runs (7 algorithms × 5 datasets × 4 $r$ values) in addition to the baseline variant. Experiments results are summarized in Table 1.5 and discussed in Section 1.4.3.

## 1.4 Results and Discussion

In this section, we show and discuss results obtained from the previous experimental evaluation. We use specific metrics that are tuned to handle outliers for an unbiased evaluation, all scores are defined by the authors in [45].

### 1.4.1 Anomaly Detection

Results in Table 1.3 support the claims in [263] and show that, as a general rule, subsampling data improves the accuracy of outlier detection algorithms, since the baseline approach (i.e., using the whole dataset) performs worse than when using coresets regardless of the technique and the considered metric. Beyond that, all studied algorithms seem to obtain representative coresets for the analysis, k-Means Coreset (KMC), SDO, and ODM slightly standing out over the rest.

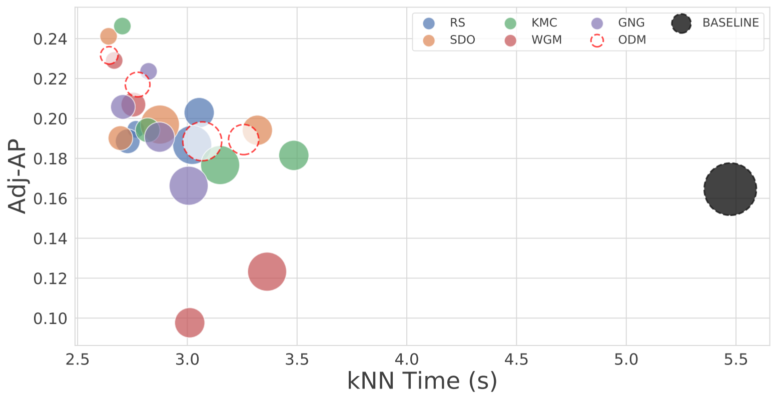
On the other hand, detection times (k-NN time) are considerably reduced when using coresets obviously because it is expected to increase the larger the dataset is. If we additionally take coreset extraction costs into account (Ext. Time), only Random Sampling (RS), SDO, and ODM seem to be light and fast enough for the undertaken task.

Weighting accuracy and time performances together, only SDO and ODM show satisfactory scores. Figure 1.9a shows a visual comparison of coresets by plotting accuracy (Adjusted Average Precision) versus analysis time (k-NN time). A comparison of coreset extraction times (Ext. Time) is shown in Figure 1.9b.

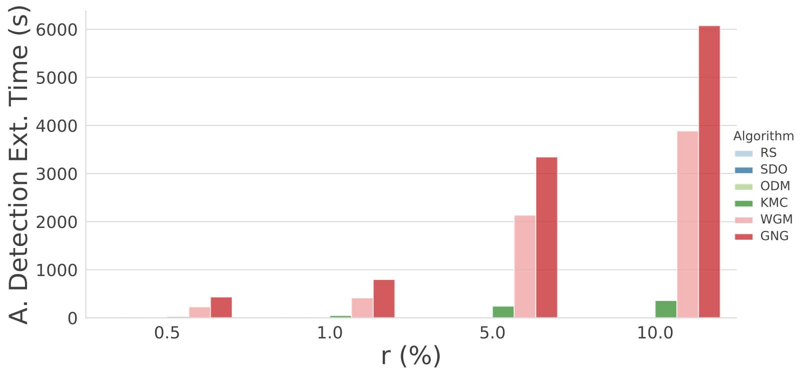
Table 1.3: Performance of detecting anomalies.

| Algorithm | r | MaxF1 | Adj-MaxF1 | P@tn | Adj-P@tn | AP | Adj-AP | ROC-AUC | Ext. Time (s) | k-NN Time (s) |
|---|---|---|---|---|---|---|---|---|---|---|
| Baseline | 0% | 0.32 ± 0.08 | 0.29 ± 0.10 | 0.14 ± 0.12 | 0.11 ± 0.10 | 0.19 ± 0.16 | 0.16 ± 0.15 | 0.83 ± 0.09 | — | 5.47 ± 5.44 |
| ODM | 0.5% | **0.45 ± 0.23** | **0.42 ± 0.24** | 0.23 ± 0.25 | 0.21 ± 0.25 | 0.26 ± 0.25 | 0.23 ± 0.24 | **0.86 ± 0.12** | 2.26 ± 2.32 | **2.64 ± 2.77** |
| RS | | 0.38 ± 0.17 | 0.35 ± 0.19 | 0.17 ± 0.16 | 0.14 ± 0.15 | 0.22 ± 0.19 | 0.19 ± 0.18 | 0.81 ± 0.12 | **0.00 ± 0.00** | 2.77 ± 2.98 |
| SDO | | 0.43 ± 0.23 | 0.40 ± 0.24 | 0.23 ± 0.26 | 0.20 ± 0.26 | **0.27 ± 0.26** | 0.24 ± 0.26 | 0.85 ± 0.12 | 0.24 ± 0.32 | 2.64 ± 2.78 |
| KMC | | 0.44 ± 0.21 | 0.41 ± 0.22 | **0.25 ± 0.27** | **0.22 ± 0.26** | 0.27 ± 0.27 | **0.25 ± 0.26** | 0.83 ± 0.12 | 23.27 ± 30.69 | 2.70 ± 2.81 |
| WGM | | 0.38 ± 0.24 | 0.35 ± 0.25 | 0.23 ± 0.29 | 0.20 ± 0.29 | 0.25 ± 0.27 | 0.23 ± 0.27 | 0.78 ± 0.13 | 228.49 ± 331.62 | 2.67 ± 2.79 |
| GNG | | 0.40 ± 0.19 | 0.37 ± 0.19 | 0.23 ± 0.25 | 0.20 ± 0.24 | 0.25 ± 0.24 | 0.22 ± 0.23 | 0.83 ± 0.12 | 433.36 ± 589.89 | 2.82 ± 2.89 |
| ODM | 1% | 0.41 ± 0.17 | 0.38 ± 0.18 | 0.20 ± 0.19 | 0.17 ± 0.18 | 0.24 ± 0.21 | 0.22 ± 0.20 | 0.84 ± 0.10 | 2.43 ± 2.44 | 2.77 ± 2.86 |
| RS | | 0.38 ± 0.17 | 0.35 ± 0.18 | 0.17 ± 0.14 | 0.14 ± 0.13 | 0.22 ± 0.18 | 0.19 ± 0.17 | 0.82 ± 0.11 | 0.00 ± 0.00 | 2.73 ± 2.81 |
| SDO | | 0.41 ± 0.18 | 0.38 ± 0.19 | 0.19 ± 0.18 | 0.16 ± 0.17 | 0.22 ± 0.19 | 0.19 ± 0.18 | 0.84 ± 0.11 | 0.43 ± 0.56 | 2.70 ± 2.82 |
| KMC | | 0.40 ± 0.18 | 0.37 ± 0.19 | 0.18 ± 0.16 | 0.15 ± 0.15 | 0.22 ± 0.19 | 0.19 ± 0.18 | 0.83 ± 0.10 | 47.00 ± 59.96 | 2.82 ± 2.95 |
| WGM | | 0.38 ± 0.23 | 0.35 ± 0.24 | 0.22 ± 0.27 | 0.19 ± 0.27 | 0.23 ± 0.27 | 0.21 ± 0.27 | 0.79 ± 0.12 | 414.47 ± 637.57 | 2.75 ± 2.89 |
| GNG | | 0.41 ± 0.18 | 0.38 ± 0.19 | 0.21 ± 0.20 | 0.18 ± 0.20 | 0.23 ± 0.21 | 0.21 ± 0.20 | 0.83 ± 0.11 | 796.74 ± 1109.50 | 2.71 ± 2.82 |
| ODM | 5% | 0.40 ± 0.16 | 0.37 ± 0.18 | 0.17 ± 0.14 | 0.14 ± 0.13 | 0.22 ± 0.18 | 0.19 ± 0.17 | 0.84 ± 0.10 | 2.41 ± 2.45 | 3.26 ± 3.29 |
| RS | | 0.39 ± 0.16 | 0.36 ± 0.18 | 0.18 ± 0.14 | 0.14 ± 0.13 | 0.23 ± 0.19 | 0.20 ± 0.18 | 0.83 ± 0.10 | 0.00 ± 0.00 | 3.05 ± 3.09 |
| SDO | | 0.38 ± 0.15 | 0.35 ± 0.17 | 0.17 ± 0.14 | 0.14 ± 0.12 | 0.22 ± 0.18 | 0.19 ± 0.17 | 0.83 ± 0.10 | 1.82 ± 2.34 | 3.32 ± 3.35 |
| KMC | | 0.41 ± 0.20 | 0.38 ± 0.21 | 0.16 ± 0.13 | 0.13 ± 0.12 | 0.21 ± 0.17 | 0.18 ± 0.16 | 0.81 ± 0.11 | 241.52 ± 325.97 | 3.49 ± 3.45 |
| WGM | | 0.26 ± 0.14 | 0.23 ± 0.16 | 0.08 ± 0.07 | 0.04 ± 0.05 | 0.13 ± 0.12 | 0.10 ± 0.10 | 0.72 ± 0.12 | 2134.30 ± 3212.13 | 3.01 ± 3.25 |
| GNG | | 0.28 ± 0.07 | 0.25 ± 0.07 | 0.16 ± 0.14 | 0.13 ± 0.12 | 0.22 ± 0.18 | 0.19 ± 0.17 | 0.81 ± 0.10 | 3344.25 ± 4983.34 | 2.87 ± 2.93 |
| ODM | 10% | 0.39 ± 0.16 | 0.36 ± 0.18 | 0.16 ± 0.13 | 0.13 ± 0.12 | 0.22 ± 0.18 | 0.19 ± 0.17 | 0.84 ± 0.10 | 2.27 ± 2.34 | 3.07 ± 3.09 |
| RS | | 0.39 ± 0.16 | 0.36 ± 0.18 | 0.16 ± 0.13 | 0.13 ± 0.12 | 0.21 ± 0.18 | 0.19 ± 0.16 | 0.84 ± 0.10 | 0.00 ± 0.00 | 3.02 ± 3.04 |
| SDO | | 0.39 ± 0.16 | 0.36 ± 0.18 | 0.16 ± 0.13 | 0.13 ± 0.12 | 0.22 ± 0.19 | 0.20 ± 0.17 | 0.83 ± 0.10 | 2.56 ± 3.33 | 2.88 ± 2.94 |
| KMC | | 0.38 ± 0.18 | 0.35 ± 0.19 | 0.16 ± 0.13 | 0.12 ± 0.11 | 0.21 ± 0.17 | 0.18 ± 0.16 | 0.81 ± 0.11 | 359.22 ± 497.37 | 3.15 ± 3.13 |
| WGM | | 0.27 ± 0.14 | 0.24 ± 0.16 | 0.11 ± 0.09 | 0.07 ± 0.09 | 0.16 ± 0.13 | 0.12 ± 0.12 | 0.72 ± 0.12 | 3884.61 ± 6105.35 | 3.36 ± 3.86 |
| GNG | | 0.27 ± 0.10 | 0.24 ± 0.09 | 0.16 ± 0.13 | 0.12 ± 0.11 | 0.19 ± 0.17 | 0.17 ± 0.15 | 0.80 ± 0.11 | 6075.36 ± 9054.07 | 3.01 ± 3.04 |

Ext. Time: Time needed to extract a coreset.



**(a)** Score achieved vs. time. Different sizes represent different $r$ values.



**(b)** Coreset extraction times.

**Figure 1.9:** Visual representation of the average anomaly detection performance.

### 1.4.2 Clustering

In the clustering experiments (results in Table 1.4), all tested algorithms show poor performances for very low $r$ (i.e., high dataset summarization). Note that the effect of $r$ is hardly generalizable as it strongly depends on the specific distributions and geometries of the data under analysis. Except for the KMC and Weighted Gaussian Mixture (WGM) algorithms, which obtain low performances even for high $r$, more quality performances are obtained the more data points we use for the coreset (i.e, higher $r$), getting progressively closer to the baseline. SDO and RS obtain almost ideal performances with $r = 10\%$.

Again, coreset extraction costs (Ext.T.) are considerably high for WGM and GNG. Weighting accuracy and time performances together, the best trade-off is obtained by SDO and ODM, or even RS. Figure 1.10a shows a scatter plot comparing accuracies (RAND index) vs analysis time (k-Means time). A comparison of coreset extraction times (Ext. Time) is displayed in Figure 1.10b.

**Table 1.4:** Clustering performance.

| **Algorithm** | **r** | **Adj. Rand Ind.** | **Silhouette Coeff.** | **Ext.T.(s)** | **kM.T.(s)** | **Lab.E.T.(s)** |
|---|---|---|---|---|---|---|
| Baseline | —— | 0.93 ± 0.03 | 0.73 ± 0.02 | —— | 1.19 ± 0.04 | —— |
| ODM | 0.5% | 0.66 ± 0.40 | 0.52 ± 0.31 | 0.14 ± 0.11 | **0.73 ± 0.44** | 0.09 ± 0.07 |
| RS | | 0.51 ± 0.35 | 0.44 ± 0.28 | **0.00 ± 0.00** | 0.74 ± 0.45 | 0.09 ± 0.06 |
| SDO | | 0.46 ± 0.34 | 0.35 ± 0.27 | **0.00 ± 0.00** | 0.68 ± 0.49 | **0.08 ± 0.07** |
| KMC | | 0.56 ± 0.40 | 0.49 ± 0.30 | 0.84 ± 0.54 | 0.80 ± 0.54 | 0.09 ± 0.07 |
| WGM | | 0.59 ± 0.41 | 0.51 ± 0.31 | 1.58 ± 2.07 | 0.78 ± 0.49 | 0.09 ± 0.07 |
| GNG | | 0.65 ± 0.40 | 0.50 ± 0.31 | 6.45 ± 5.47 | 0.82 ± 0.53 | 0.10 ± 0.08 |
| ODM | 1% | 0.90 ± 0.04 | 0.71 ± 0.02 | 0.21 ± 0.13 | 1.06 ± 0.15 | 0.11 ± 0.04 |
| RS | | 0.84 ± 0.08 | 0.67 ± 0.08 | **0.00 ± 0.00** | 1.03 ± 0.14 | 0.11 ± 0.04 |
| SDO | | 0.75 ± 0.22 | 0.59 ± 0.18 | 0.01 ± 0.00 | 0.95 ± 0.27 | 0.11 ± 0.06 |
| KMC | | 0.69 ± 0.21 | 0.63 ± 0.08 | 1.17 ± 0.26 | 0.99 ± 0.09 | 0.12 ± 0.06 |
| WGM | | 0.76 ± 0.22 | 0.67 ± 0.05 | 2.51 ± 1.96 | 1.09 ± 0.25 | 0.12 ± 0.06 |
| GNG | | 0.88 ± 0.05 | 0.70 ± 0.05 | 10.13 ± 6.38 | 1.12 ± 0.28 | 0.11 ± 0.06 |
| ODM | 5% | 0.90 ± 0.06 | 0.70 ± 0.06 | 0.20 ± 0.10 | 1.02 ± 0.07 | 0.11 ± 0.04 |
| RS | | 0.90 ± 0.05 | 0.71 ± 0.03 | **0.00 ± 0.00** | 1.02 ± 0.07 | 0.11 ± 0.03 |
| SDO | | 0.90 ± 0.04 | 0.71 ± 0.03 | 0.04 ± 0.03 | 1.03 ± 0.06 | 0.11 ± 0.04 |
| KMC | | 0.68 ± 0.23 | 0.59 ± 0.15 | 2.02 ± 1.03 | 0.99 ± 0.15 | 0.11 ± 0.05 |
| WGM | | 0.67 ± 0.24 | 0.57 ± 0.19 | 12.64 ± 10.52 | 1.09 ± 0.18 | 0.12 ± 0.05 |
| GNG | | 0.84 ± 0.10 | 0.69 ± 0.06 | 23.09 ± 17.53 | 1.11 ± 0.30 | 0.11 ± 0.06 |
| ODM | 10% | 0.89 ± 0.12 | 0.70 ± 0.08 | 0.19 ± 0.08 | 1.01 ± 0.03 | 0.11 ± 0.04 |
| RS | | **0.92 ± 0.03** | **0.72 ± 0.02** | **0.00 ± 0.00** | 1.00 ± 0.04 | 0.11 ± 0.04 |
| SDO | | **0.92 ± 0.03** | **0.72 ± 0.02** | 0.07 ± 0.05 | 1.00 ± 0.03 | 0.12 ± 0.05 |
| KMC | | 0.66 ± 0.25 | 0.57 ± 0.16 | 3.01 ± 1.62 | 0.97 ± 0.05 | 0.10 ± 0.04 |
| WGM | | 0.57 ± 0.24 | 0.50 ± 0.23 | 25.30 ± 18.58 | 1.03 ± 0.12 | 0.11 ± 0.04 |
| GNG | | 0.79 ± 0.21 | 0.65 ± 0.11 | 34.18 ± 26.41 | 1.09 ± 0.22 | 0.11 ± 0.06 |

Ext.T.: Time needed to build a coreset.
kM.T.: Time needed to build a k-Means model.
Lab.E.T.: Time needed to extend labels.

### 1.4.3 Supervised Classification

Similarly to the unsupervised classification experiments, in the supervised classification framework higher values of $r$ tend to improve performances and get closer to the baseline (see experiment results in Table 1.5). It is worth noting that the performance reduction when using coresets compared with the baseline case is considerable. Nonetheless, the best performances are obtained by ODM, KMC, and RS. Even in spite of the fact that analysis run-times (k-NN time) are not discriminating and more or less equally demanding for all cases, KMC is still a bit slower than other algorithms. On the other hand, if we examine coreset extraction costs (Ext. Time), there are two clearly separable groups: (a) light extraction: RS, ODM, and SDO; and (b) heavy extraction: KMC, WGM, GNG, and CNN. Weighting accuracy and time performances together, ODM and RS offer the best solutions with affordable complexities.

The CNN additionally appears in this set of experiments as it was specifically proposed for alleviating the computational costs of k-nn-based classification. However, it is a

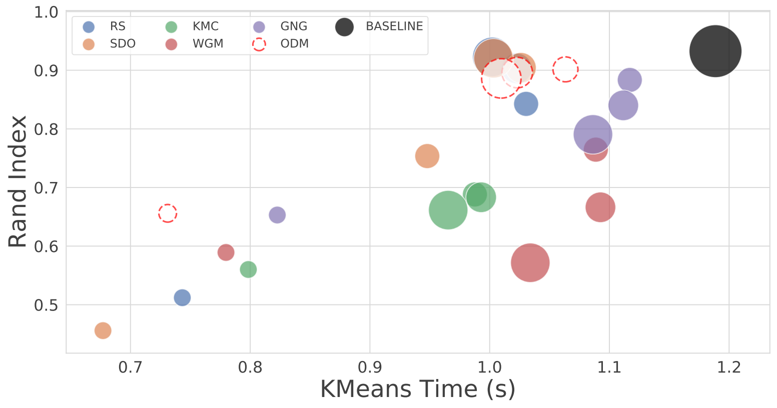
**Table 1.5:** Supervised learning performance.

| Algorithm | r | Accuracy | Mi. Prec. | Ma. Prec. | Mi. Rec. | Ma. Rec. | Ext. Time (s) | k-NN Time (s) |
|---|---|---|---|---|---|---|---|---|
| Baseline | — | $0.94 \pm 0.09$ | $0.94 \pm 0.09$ | $0.88 \pm 0.09$ | $0.94 \pm 0.09$ | $0.84 \pm 0.10$ | — | $0.29 \pm 0.28$ |
| ODM | | $0.85 \pm 0.18$ | $0.85 \pm 0.18$ | $0.56 \pm 0.30$ | $0.85 \pm 0.18$ | $0.56 \pm 0.31$ | $0.63 \pm 0.30$ | $0.12 \pm 0.06$ |
| RS | | $0.83 \pm 0.19$ | $0.83 \pm 0.19$ | $0.60 \pm 0.30$ | $0.83 \pm 0.19$ | $0.52 \pm 0.32$ | $\mathbf{0.00 \pm 0.00}$ | $0.12 \pm 0.06$ |
| SDO | | $0.75 \pm 0.24$ | $0.75 \pm 0.24$ | $0.46 \pm 0.29$ | $0.75 \pm 0.24$ | $0.51 \pm 0.33$ | $0.02 \pm 0.01$ | $0.11 \pm 0.05$ |
| KMC | 0.5% | $0.82 \pm 0.19$ | $0.82 \pm 0.19$ | $0.55 \pm 0.34$ | $0.82 \pm 0.19$ | $0.46 \pm 0.27$ | $4.88 \pm 3.40$ | $0.19 \pm 0.08$ |
| WGM | | $0.65 \pm 0.39$ | $0.65 \pm 0.39$ | $0.54 \pm 0.38$ | $0.65 \pm 0.39$ | $0.45 \pm 0.34$ | $6.52 \pm 7.34$ | $\mathbf{0.09 \pm 0.07}$ |
| GNG | | $0.82 \pm 0.19$ | $0.82 \pm 0.19$ | $0.57 \pm 0.32$ | $0.82 \pm 0.19$ | $0.48 \pm 0.25$ | $31.30 \pm 16.28$ | $0.12 \pm 0.06$ |
| CNN | | $0.63 \pm 0.29$ | $0.63 \pm 0.29$ | $0.41 \pm 0.29$ | $0.63 \pm 0.29$ | $0.50 \pm 0.23$ | $65.94 \pm 115.28$ | $0.11 \pm 0.05$ |
| ODM | | $0.88 \pm 0.17$ | $0.88 \pm 0.17$ | $0.63 \pm 0.26$ | $0.88 \pm 0.17$ | $0.59 \pm 0.28$ | $0.73 \pm 0.44$ | $0.14 \pm 0.09$ |
| RS | | $0.85 \pm 0.19$ | $0.85 \pm 0.19$ | $0.62 \pm 0.27$ | $0.85 \pm 0.19$ | $0.56 \pm 0.29$ | $0.01 \pm 0.01$ | $0.14 \pm 0.09$ |
| SDO | | $0.83 \pm 0.24$ | $0.83 \pm 0.24$ | $0.60 \pm 0.29$ | $0.83 \pm 0.24$ | $0.57 \pm 0.30$ | $0.04 \pm 0.02$ | $0.11 \pm 0.05$ |
| KMC | 1% | $0.86 \pm 0.18$ | $0.86 \pm 0.18$ | $0.62 \pm 0.29$ | $0.86 \pm 0.18$ | $0.54 \pm 0.30$ | $5.59 \pm 3.79$ | $0.19 \pm 0.09$ |
| WGM | | $0.67 \pm 0.39$ | $0.67 \pm 0.39$ | $0.54 \pm 0.38$ | $0.67 \pm 0.39$ | $0.50 \pm 0.35$ | $12.16 \pm 13.37$ | $0.10 \pm 0.08$ |
| GNG | | $0.84 \pm 0.17$ | $0.84 \pm 0.17$ | $0.60 \pm 0.30$ | $0.84 \pm 0.17$ | $0.53 \pm 0.25$ | $41.26 \pm 22.74$ | $0.12 \pm 0.06$ |
| CNN | | $0.68 \pm 0.29$ | $0.68 \pm 0.29$ | $0.44 \pm 0.26$ | $0.68 \pm 0.29$ | $0.53 \pm 0.21$ | $66.64 \pm 116.62$ | $0.11 \pm 0.05$ |
| ODM | | $0.90 \pm 0.14$ | $0.90 \pm 0.14$ | $0.67 \pm 0.25$ | $0.90 \pm 0.14$ | $0.62 \pm 0.25$ | $0.64 \pm 0.31$ | $0.14 \pm 0.08$ |
| RS | | $0.89 \pm 0.16$ | $0.89 \pm 0.16$ | $0.75 \pm 0.26$ | $0.89 \pm 0.16$ | $0.62 \pm 0.26$ | $\mathbf{0.00 \pm 0.00}$ | $0.14 \pm 0.08$ |
| SDO | | $0.64 \pm 0.37$ | $0.64 \pm 0.37$ | $0.50 \pm 0.41$ | $0.64 \pm 0.37$ | $0.44 \pm 0.32$ | $0.18 \pm 0.14$ | $0.12 \pm 0.06$ |
| KMC | 5% | $0.89 \pm 0.15$ | $0.89 \pm 0.15$ | $0.66 \pm 0.25$ | $0.89 \pm 0.15$ | $0.60 \pm 0.23$ | $10.38 \pm 6.30$ | $0.21 \pm 0.11$ |
| WGM | | $0.63 \pm 0.37$ | $0.63 \pm 0.37$ | $0.53 \pm 0.35$ | $0.63 \pm 0.37$ | $0.56 \pm 0.35$ | $54.34 \pm 58.35$ | $0.11 \pm 0.10$ |
| GNG | | $0.89 \pm 0.13$ | $0.89 \pm 0.13$ | $0.67 \pm 0.25$ | $0.89 \pm 0.13$ | $0.61 \pm 0.25$ | $106.06 \pm 69.87$ | $0.14 \pm 0.08$ |
| CNN | | $0.85 \pm 0.15$ | $0.85 \pm 0.15$ | $0.67 \pm 0.16$ | $0.85 \pm 0.15$ | $0.69 \pm 0.23$ | $66.50 \pm 116.09$ | $0.14 \pm 0.07$ |
| ODM | | $\mathbf{0.91 \pm 0.13}$ | $\mathbf{0.91 \pm 0.13}$ | $\mathbf{0.78 \pm 0.24}$ | $\mathbf{0.91 \pm 0.13}$ | $0.64 \pm 0.24$ | $0.65 \pm 0.31$ | $0.16 \pm 0.10$ |
| RS | | $0.90 \pm 0.15$ | $0.90 \pm 0.15$ | $0.77 \pm 0.23$ | $0.90 \pm 0.15$ | $\mathbf{0.64 \pm 0.25}$ | $\mathbf{0.00 \pm 0.00}$ | $0.15 \pm 0.09$ |
| SDO | | $0.63 \pm 0.36$ | $0.63 \pm 0.36$ | $0.54 \pm 0.35$ | $0.63 \pm 0.36$ | $0.49 \pm 0.30$ | $0.38 \pm 0.30$ | $0.14 \pm 0.08$ |
| KMC | 10% | $0.90 \pm 0.13$ | $0.90 \pm 0.13$ | $0.77 \pm 0.25$ | $0.90 \pm 0.13$ | $0.61 \pm 0.22$ | $14.65 \pm 8.91$ | $0.23 \pm 0.13$ |
| WGM | | $0.60 \pm 0.36$ | $0.60 \pm 0.36$ | $0.53 \pm 0.34$ | $0.60 \pm 0.36$ | $0.57 \pm 0.35$ | $104.27 \pm 112.55$ | $0.13 \pm 0.12$ |
| GNG | | $\mathbf{0.91 \pm 0.13}$ | $\mathbf{0.91 \pm 0.13}$ | $0.76 \pm 0.25$ | $0.91 \pm 0.13$ | $0.63 \pm 0.21$ | $173.76 \pm 122.99$ | $0.16 \pm 0.10$ |
| CNN | | $0.75 \pm 0.30$ | $0.75 \pm 0.30$ | $0.55 \pm 0.27$ | $0.75 \pm 0.30$ | $0.63 \pm 0.22$ | $66.16 \pm 115.79$ | $0.15 \pm 0.09$ |

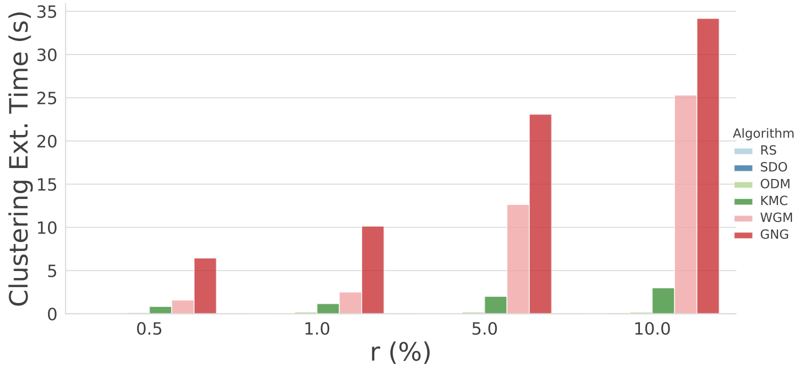
**Ext.T.**: Time needed to build a coreset.

**Ma.**: Macro.

**Mi.**: Micro.



**(a)** Score achieved vs time. Different sizes represent different $r$ values.



**(b)** Coreset extraction times.

**Figure 1.10:** Visual representation of the average clustering performance.

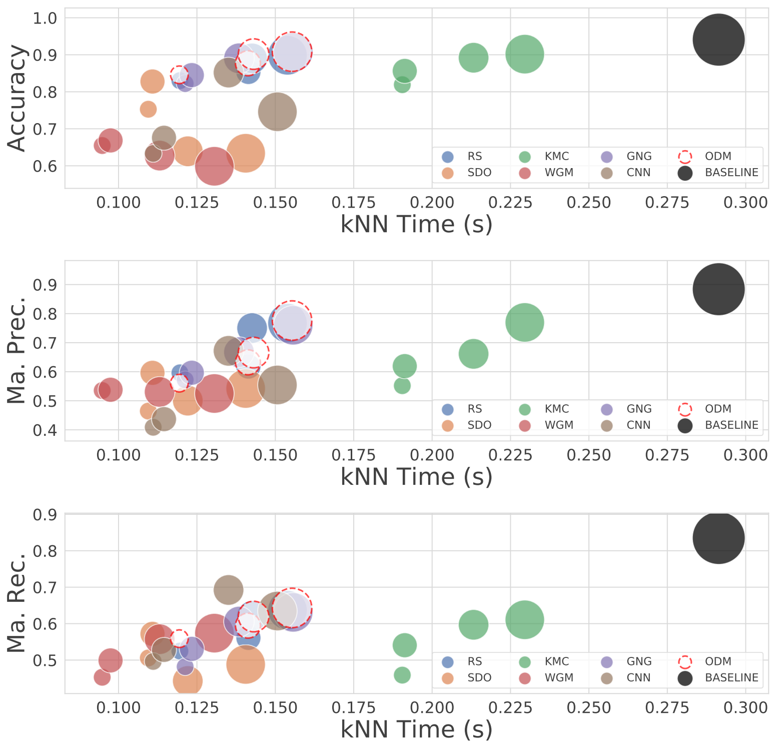
heavy algorithm in terms of extraction costs when compared to ODM or SDO, and accuracy performances are in line or below most of the competitors. Note that, in its original implementation, CNN automatically establishes the size of the coreset according to dataset properties and cannot be externally implemented. This fact might be affecting its capability of obtaining better performances.

Alike previous experiments, Figure 1.11a compares coresets by using classification metrics (Accuracy, Macro Precision, Macro Recall) vs analysis time (k-NN time). A comparison of coreset extraction times (Ext. Time) is displayed in Figure 1.11b.

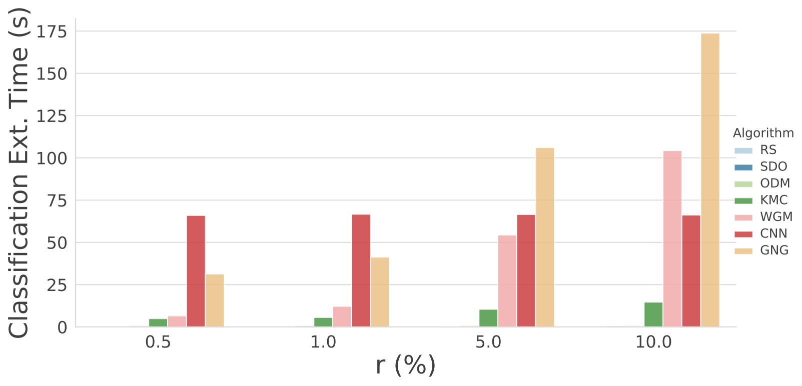
## 1.4.4 Results Overview

Evaluating all experiments together, ODM stands out as it is the only particularly stable method (i.e, its performance are among the top ones regardless of the application) and keeps considerable low time-complexity costs.

To confirm the statistically significant differences among coreset extracting algorithms

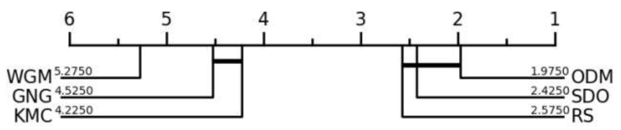


**(a)** Score achieved vs time. Different sizes represent different $r$ values.

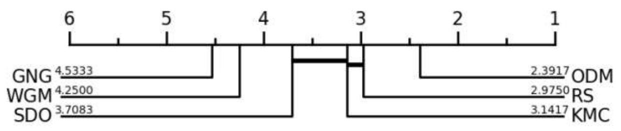


**(b)** Coreset extraction times.

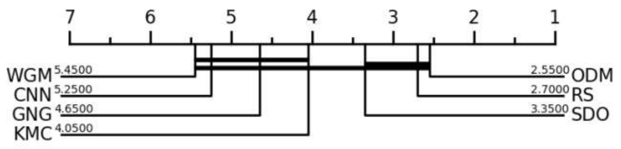
**Figure 1.11:** Visual representation of the average SL performance.



**(a)** Anomaly detection.



**(b)** Clustering.



**(c)** Supervised learning.

**Figure 1.12:** Critical Difference Diagrams. The methods are ordered in such a way that the best one is the rightmost one. Methods whose difference is not shown to be significant in the tests are joined with a thick line.

under test, we use Critical Difference Diagrams (CDDs) [69, 124], which are based on the Wilcoxon signed-rank test [245]. Figure 1.12 shows the CDD for our three experimental applications: anomaly detection, clustering, and supervised classification. For the calculation of the CDD, we combine all metrics presented in Table 1.3, Table 1.4 and Table 1.5 after z-scoring them to make them comparable and addable. Thus, obtaining one summary metric for each application. In the CDD, methods are ordered to keep the best ones on the right side of the diagram. Methods whose different performances are not deemed significant are joined by thick horizontal lines. CDDs confirm the preponderance of ODM over competitors regardless of application, followed by RS and SDO.

## 1.5 Conclusion

In this chapter, we present ODM, a lightweight algorithm for the extraction of low-density data models (data summaries, coresets) in log-linear times that helps improving the quality and speed of subsequent analysis. We test ODM and compare it with state-of-the-art alternatives by using multiple datasets in fundamental machine learning areas, namely: anomaly detection, clustering, and supervised classification. Obtained results place ODM as the most stable option and the best trade-off between time-complexity and performance accuracy. ODM is strongly recommended for alleviating computational demands when analyzing large datasets such as network traffic traces, which works by only retaining the data's core structure. Future research in this direction includes using ODM as an under-sampling method for cases where classes show strong imbalance (e.g., attack detection in network traffic); applying ODM for streaming data, where it is necessary to keep a model of the data up to date (therefore, able to evolve - see Part II - Chapter 2 - Section 2.4.3), but at the same time, preventing that the accumulation of new samples deteriorate the model in use. Finally, evaluate the potential of ODM to discover emerging clusters in streaming applications and differentiate them quickly and efficiently from simple anomalies.

CHAPTER 2

# Generative Networks for Flow and Packet Generation

***Notice of adoption from previous publications - 7***
*Parts of this section's content have been presented as a poster:*
*Fares Meghdouri, Thomas Schmied, Thomas Gärtner, Tanja Zseby: Controllable Network Data Balancing With GANs. Deep Generative Models and Downstream Applications Workshop - NIPS 2021*[a].

[a]dgms-and-applications.github.io/2021/

Despite the fact that enormous amounts of network data are generated every day as a result of the Internet's pervasiveness in our daily lives, the amount of network traffic that can be used for research purposes is comparatively limited. In reality, the few publicly available datasets for intrusion detection have a number of flaws, including artifacts if the traffic is artificially generated [135]. Nonetheless, future network anomaly detection research is limited by the availability of large and diverse network traffic datasets, besides, there has been a trend towards ML-based IDSs that rely heavily on large-scale datasets that contain a mixture of network traffic [184].

## 2.1 Introduction

In this chapter we present a framework based on GANs to generate and augment malicious network traffic samples in order to make anomaly detection methods more robust. Effectively, we do not only generate similar samples, but also samples with specific characteristics that necessitates the use of a condition-based generation process. We compare the performance of four established GAN architectures: Conditional GAN (CGAN) [173], Wasserstein GAN (WGAN) [16], Auxiliary-Classifier GAN (AC-GAN) [183], and Wasserstein GAN with Auxiliary Classifier (AC-WGAN). We also experimented with Wasserstein GAN with Gradient Penalty (WGAN-GP) [100], but results were inconclusive. All of the preceding architectures support a defined input vector that is called the "condition" for controlling the generation process.

We propose two methods for controlling attack generation. First, as is standard practice, we set conditions on the network attack type in a binary format (malicious/benign). This enables the generation of flows that resemble a specific attack type in the training set, such as DDoS attacks. However, this method only allows for coarse control. As an alternative, the second approach aims at a finer-grained control over the generation process by setting conditions on a set of the most representative and intuitive features (for example, "Flow Duration" or "Mean Packet Length" - here the notation of features differs from previous chapters for readability reasons). This doesn't only allows for the generation of an attack samples of a specific type, but also of an attack that should, for example, have a short "Flow Duration" and/or a high "Mean Packet Length". In general, this method allows to diversify the generation process thus, increasing the robustness of the IDS when used.

In short, we make the following contributions in this chapter:

- We propose an architecture for balancing network traffic datasets with GANs and compare it to Synthetic Minority Oversampling Technique (SMOTE).
- We propose and evaluate two control techniques for augmenting network traffic datasets.
- We propose an approach that, combined with GANs, allows for fine-grained control over the generation process.
- We propose a fine-grained approach to generate custom packets sequences instead of flows.

## 2.2 Proposal: FlowGan

In this section, we describe our proposal. First, we provide a brief summary on aggregated flows extracted from the CICIDS2017 dataset [220] as a reminder. Then we describe the model's architecture, as well as our two conditioning approaches. Finally, we describe our evaluation strategy for both data augmentation and controlled attack sample generation.

### 2.2.1 Used Data

Used multiple times throughout this thesis, the CICIDS2017 dataset is a well-known collection of network traffic containing 14 different types of attacks as well as a variety of normal traffic protocols and services. It has recently been also extensively used in network traffic analysis related research[238, 204, 135, 166]. In Table 2.1 we present the number of samples and proportion of each class in the dataset. The dataset is highly

| **Class** | **Samples** | **Proportion (%)** |
|---|---|---|
| *BENIGN* | *2273097* | *80.30037* |
| DoS → Hulk | 231073 | 8.16298 |
| Port Scan | 158930 | 5.61443 |
| DDoS | 128027 | 4.52273 |
| DoS → GoldenEye | 10293 | 0.36361 |
| Patator → FTP | 7938 | 0.28042 |
| Patator → SSH | 5897 | 0.20832 |
| DoS → slowloris | 5796 | 0.20475 |
| DoS → Slowhttptest | 5499 | 0.19426 |
| Bot | 1966 | 0.06945 |
| Web Attack → Brute Force | 1507 | 0.05324 |
| Web Attack → XSS | 652 | 0.02303 |
| Infiltration | 36 | 0.00127 |
| Web Attack → Sql Injection | 21 | 0.00074 |
| Heartbleed | 11 | 0.00039 |

**Table 2.1:** Number of samples per class in CICIDS2017

imbalanced: benign activity accounts for 80% of the traffic, the first three attack classes account for 18%, and the rest account for only 2% of all samples in the dataset. This is reflected in results presented in the following sections, as the proposed framework performs well on over-represented classes but poorly on under-represented ones, such as *Heartbleed* or *Sql Injection*. Nonetheless, this is not surprising and further discussed in Section 2.4.

Each sample in the dataset represents a communication flow with 85 statistical features (the complete list can be found in the respective paper [220] and represents to some extent a combination of both CAIA and Consensus feature vectors discussed in Part I/Chapter 4). We consider the "Flow ID", "Source IP", "Destination IP", "Source Port", "Destination Port", "Protocol", and "Timestamp" as traffic identifiers rather than statistical features and therefore we drop them during the generation process. If the "Destination Port" or the "Protocol" are required during analysis after the generation process, they can simply be prepended based on the sample class because each class often has a unique value for these features.

### 2.2.2 GANs for Data Balancing: Proposed Architecture

We use a set of GAN architectures for both sample generation and augmentation. The architectures are: CGAN, WGAN, AC-GAN and AC-WGAN as discussed in the beginning of this chapter. Figure 2.1 illustrates the proposed model on a conceptual level. Overall, there are two central components: the generator and the discriminator which are both

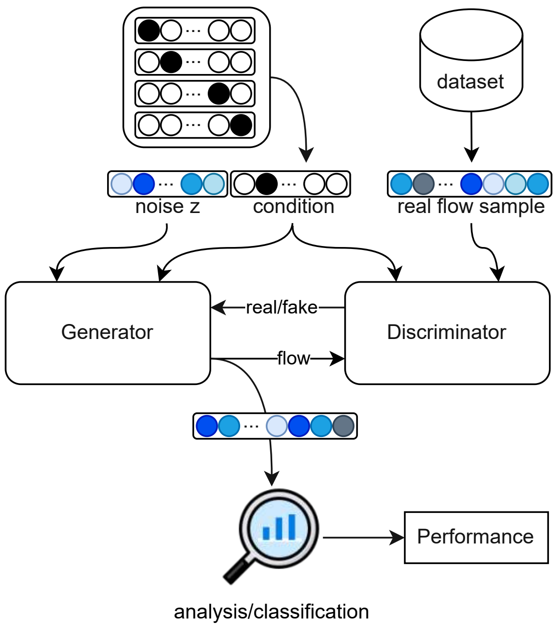


**Figure 2.1:** Proposed architecture.

FCNNs. The generator produces increasingly more realistic traffic flow samples whereas the discriminator evaluates whether the given samples are real or fake i.e., come from the same distribution as the original samples or not. Hence, the generator and discriminator are, in a way, competing against each other. As one component improves the other keeps up, and vice-versa. This is the fundamental idea of GANs [97]. Among the different architectures discussed above, we keep the same network size and we only alter the optimizers. Before passing them into the network, we apply Min-Max scaling on all samples.

**Generator**. In order to generate new (attack) samples, the generator is given two inputs: a noise vector $\mathbf{z}$ concatenated with an additional condition vector (discussed in what follows). The noise vector $\mathbf{z}$ is a 128-dimensional random vector sampled from a normal distribution ($\mu = 0$, $\sigma^2 = 1$) that is generated anew with each forward pass. The generator is composed of two 256 neuron layers and one 512 neuron layer. We use batch normalization [123] between layers to re-scale batches and make the network training more stable (this is better elaborated in Chapter 4), as well as *Leaky Relu* activations [162] ($\alpha = 0.1$), except at the final layer, where we use a linear activation.

**Discriminator.** During training, the discriminator determines whether attack samples generated by the generator are real or fake. To learn the ability of distinguishing real samples from fake ones, it also receives real attack samples obtained from the dataset. Therefore, the discriminator is passed two inputs: a similar (attack) sample to the one that has to be generated, picked from the original dataset and an additional condition. Depending on the used GAN architecture, there are a few differences. For example, with

AC-GAN the discriminator doesn't only predict whether the input is real or fake but also what class it belongs to. The discriminator has a similar architecture to the generator, but we use no batch normalization.

**Training.** To compute the loss of the generator and the two losses (real and fake samples) of the discriminator, we use a sigmoid activation function followed by the Binary Cross Entropy (BCE) loss. Furthermore, we use *Adam* for optimization [138] ($lr = 0.0002$) or *RMSprop* [114] for the WGAN case with the same $lr$.

### 2.2.3 Coarse control

In order to generate a sample of a specific attack, we represent the condition vector shown in in Figure 2.1 as a static one-hot vector that encodes the attack class. As there are 14 attack classes, the condition vector is of dimension 14. For CGAN and WGAN this condition vector is provided for both the generator and the discriminator. However, for models that additionally employ an auxiliary classifier within the discriminator, only the generated "real" sample is fed into the discriminator, otherwise, it would be simple to predict the right class and the additional cross entropy loss on the attack class would be useless. This setup makes it possible to not only generate attack samples, but samples of a particular attack class.

### 2.2.4 Fine-grained control

While the per-class conditional approach above allows to generate attacks of a particular attack class, it only enables coarse control (i.e., generated samples exhibit the same statistical characteristics as the ones provided during training). Therefore, we propose a dynamic condition vector that aims at a more fine-grained control over the generation process, by setting conditions on a subset of eleven discriminators and pushing the GAN to interpolate the entire output vector therefore, generating the output space out of a subset of the input space represented by a condition. The features selected as input condition are shown in Table 2.2. These features are selected based on their relevancy (feature importance) extracted from a RF model trained on the original data. Instead of constructing a condition vector using the plain feature values, we encode them in such a way that they are easily adjustable during the generation process as shown in the same table.

Overall, this results in a 25-dimensional binary condition vector ($3\times7+4 = 25$ dimensions) for each training sample as illustrated in Figure 2.2. For example, we can specify that attack "X" with a short "Flow Duration" and high "Mean Packet Length" should be generated. However, if the goal is simply to augment a certain predefined class, we can achieve so by computing the mean value across all samples of a given attack sample and then extract the respective dynamic condition vector to be used as condition. In contrary to the static class condition vector, generating the training set in this case is not trivial. We pick at most 10k random samples from each class then construct the

| **Feature** | **Condition on Value (levels)** |
|---|---|
| Flow Duration<br>Total Backward Packets<br>Total Forward Packets<br>Mean Packet Length<br>Bytes/s<br>Minimum IAT<br>Maximum IAT | Low/Mid/High |
| PSH Flags<br>SYN Flags<br>RST Flags<br>ACK Flags | Flags/No Flags |

**Table 2.2:** Dynamic features chosen for the condition vector. The naming convention changes here compared to Table 1.2 in Part II - Chapter 1 because a different extraction tool is used however, the same features are present in both tables and the 1-to-1 relation can be easily found. The levels LOw/Mid/High are derived from the quantiles of the original distribution.

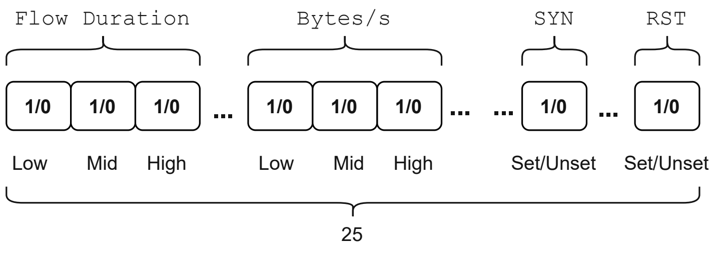


**Figure 2.2:** Dynamic Condition Vector.

dynamic condition vector for each sample, as explained below. The resulting subset is then used as a training set.

All three-level features are originally continuous. In order to convert them into a three level binary vector, each feature value is binned into 3 buckets: low, mid and high. For each feature in the training dataset the 33rd and the 66th quantiles ($Q_{33}$ and $Q_{66}$) are computed and the encoded input vector ($\mathbf{I}_{x,f}$) of a feature $f$ of sample $\mathbf{x}$ is chosen such that:

$$\mathbf{I}_{x,f} = \begin{cases} [1,\,0,\,0],(low) & \text{if} \quad \mathbf{x}_f < Q_{33} \\ [0,\,1,\,0],(mid) & \text{if} \quad Q_{33} \leq \mathbf{x}_f < Q_{66} \\ [0,\,0,\,1],(high) & \text{if} \quad Q_{66} \leq \mathbf{x}_f \end{cases} \tag{2.2.1}$$

The flag-features however, are binary and determine whether a flag is set at least once in the flow or not. Therefore, we simply append them with the condition vector as a singular binary dimension.

## 2.3 Evaluation

Often, GANs are used to generate 2D data (i.e. images) which makes the (visual) evaluation and inspection easier. However, when dealing with tabular data the evaluation is not as straightforward and generated samples cannot be visually validated [35]. Therefore, our approach requires additional methods to evaluate the quality of the generated samples. We evaluate the validity of the generated network flows on four levels to ensure that the previous statement holds true. Validation steps include: 1) Attack validity by checking if specific feature values are within the allowed ranges (e.g., a valid protocol number, a non-negative packet length etc.), 2) Application of an external classifier (to check whether the generated samples are detected as attack), 3) Statistical tests on the distribution of generated features in order to compare them to the original distributions, and 4) Distance measures between generated and real attacks. We note here that recent similar work on GANs for network traffic generation uses the same approaches for evaluation [203, 202, 50].

### 2.3.1 Attack Validity

One aspect to be considered is ensuring that generated samples represent genuine attack flows. Unfortunately, GANs can interpolate or extrapolate distributions if no penalty is used hence, are blind to the generated values and their meaning. To solve this, we set rules to filter out all samples with inconsistencies in the feature values, such as samples where the "Maximum IAT" is less than the "Minimum IAT". The rules for each feature that exhibits multiple statistical operators are set as follows:

- Max(X) > Min(X)
- Max(X) > Mean(X) > Min(X)

### 2.3.2 External Classifier

After cleaning generated data, we assess its quality using an external RF classifier trained on the original dataset (excluding normal traffic). The idea here is that generated network flows should produce high detection ratios when passed to the IDS. We choose RF because it trains quickly and produces the highest classification scores, as presented by the dataset authors [220]. To avoid over-fitting, we perform a 5-fold cross-validation and report the accuracy-, precision-, recall-, and F1-scores (marco and weighted) in the comparison.

### 2.3.3 Statistical Test

In addition, we run statistical tests to compare generated data distributions to the real ones. The underlying assumption of this evaluation method is that for generated samples to be realistic, their feature distributions should be similar to real ones. As a result, we apply the two-sample t-test with significance level $\alpha = 5\%$ [223] to each feature in a batch of generated and real attack samples. The test considers a feature to be "real" if its distribution is similar to the original one and their means are approximately the same. In the best-case scenario, all distributions are not significantly different when compared pairwise, and thus all features are considered real.

### 2.3.4 Distance Measures

We compute the average Euclidean distance between generated samples and the original class centroid. Smaller distances represent samples near the original class mean and vice versa.

### 2.3.5 Output Control

We also assess how well the generation process can be controlled under dynamic conditions. We accomplish this by varying one feature at a time and observing the effect on the output. However, because the input condition contains both continuous and binary features, we group them into two categories: three-level features and binary features.

**Three-level features.** We change the value of each feature three times, setting it to low $[1, 0, 0]$, mid $[0, 1, 0]$, or high $[0, 0, 1]$ and then generate 1000 samples of each attack class then check whether the average $\mu$ over all samples meets certain criteria (described bellow) by comparing it to the means of the real samples $\mu_{low}$, $\mu_{mid}$ and $\mu_{high}$. If the criteria is met, an initially zeroed quantity called the "success score" is incremented by 1. As an example:

- Set the "Mean Packet Length" to a low value $[1, 0, 0]$,
- generate 1000 DDoS samples,
- compute the average value of the "Mean Packet Length" at the output,
- increment the "success score" based on the following conditions:
  - ➔ +1 **if** *low* is selected **and** $\mu < \mu_{mid} < \mu_{high}$,
  - ➔ +1 **if** *mid* is selected **and** $\mu_{low} < \mu < \mu_{high}$,
  - ➔ +1 **if** *high* is selected **and** $\mu_{low} < \mu_{mid} < \mu$,
  - ➔ +0 otherwise.

To make the success score more understandable, we normalize it by the total number of tests: number of attack classes × number of features × number of levels ($14 \times 7 \times 3 = 294$), this converts it into a "success rate".

**Binary features.** For transport layer flags, we again generate 1000 samples per class and flag, then compute the number of matches between the original binary feature and the supposedly (similar) generated feature, in this case the accuracy score is a suitable tool to perform this calculation.

| Metric | AC-GAN | AC-WGAN | CGAN | WGAN | Dynamic-CGAN | SMOTE |
|---|---|---|---|---|---|---|
| Accuracy | **0.86±0.02** | 0.72±0.03 | 0.84±0.01 | 0.55±0.10 | 0.83±0.01 | 0.67±0.00 |
| Macro f1 | 0.19±0.00 | 0.27±0.02 | 0.48±0.13 | 0.15±0.05 | **0.52±0.03** | 0.38±0.04 |
| Macro precision | 0.19±0.00 | 0.31±0.01 | 0.45±0.09 | 0.14±0.04 | **0.56±0.08** | 0.39±0.06 |
| Macro recall | 0.22±0.00 | 0.30±0.02 | **0.57±0.11** | 0.18±0.08 | 0.52±0.06 | 0.42±0.09 |
| Weighted f1 | 0.83±0.02 | 0.71±0.02 | **0.85±0.02** | 0.53±0.12 | 0.84±0.01 | 0.79±0.02 |
| Weighted precision | 0.78±0.01 | 0.71±0.00 | **0.88±0.06** | 0.57±0.11 | 0.82±0.02 | 0.78±0.02 |
| Weighted recall | **0.87±0.01** | 0.72±0.01 | 0.84±0.01 | 0.56±0.10 | 0.85±0.02 | 0.80±0.06 |

**Table 2.3:** Achieved metrics for 10k sample per class over two trials.

## 2.4 Results and Discussion

In this section, we present and discuss the obtained results. The source code as well as the trained models are publicly available[1]. During training, we ensure that all architectures are trained for the same number of epochs (150).

### 2.4.1 Data Augmentation

As discussed in the previous section, we perform a variety of tests with 10k generated samples of each class to assess the quality of generated samples when augmenting the training dataset or balancing the minority class. Furthermore, we compare evaluation scores obtained using four GAN architectures in addition to those obtained with SMOTE [51] which is a widely used de-facto technique for balancing datasets.

To begin with, we show post-classification scores obtained with the RF classifier in Table 2.3. High detection scores represent fake samples that were correctly classified as attacks by the IDS trained on the original set. We observe that AC-GAN has the highest accuracy, whereas CGAN has the highest macro and weighted f1 scores with its two condition vectors. This demonstrates that the CGAN architecture can generate distinguishable attack types with high detection scores. SMOTE, on the other hand, performed comparably poor but still superior to WGAN and AC-WGAN.

Table 2.4 shows the average distance to the original class mean across all classes. Lower numbers indicate a shorter distance. Unsurprisingly, SMOTE obtains the lowest (best) score because its underlying mechanism generates samples interpolating existing sample pairs; thus, all fake samples are relatively close to the centroid of each cluster. CGAN obtains the next best score with both condition vectors, indicating a high similarity between artificially-generated and real samples.

Next, we show the estimated average number of “real” features in Table 2.5. The metric is derived using the t-test discussed in Section 2.3.3 which tells whether distribution means of the generated samples match the real ones. We obtain a maximum number of 78 match (78 features have a relatively similar mean). We again observe that SMOTE has the highest score because it is an interpolation tool. However, using dynamic conditioning

[1] `github.com/CN-TU/controllable-network-data-balancing-with-gans`

with CGAN, we still achieve a similar score. This shows that on average, generated samples exhibit a similar distribution as the real ones.

| Architecture | Distance |
|---|---|
| AC-GAN | 0.40±0.08 |
| AC-WGAN | 0.41±0.08 |
| CGAN | 0.35±0.05 |
| WGAN | 0.49±0.21 |
| Dynamic-CGAN | 0.39±0.06 |
| SMOTE | **0.24±0.10** |

**Table 2.4:** Average distance to the class mean over two trials.

| Architecture | Real Features |
|---|---|
| AC-GAN | 19.7±1.5 |
| AC-WGAN | 28.1±0.4 |
| CGAN | 25.0±4.3 |
| WGAN | 25.6±0.1 |
| Dynamic-CGAN | 41.8±1.0 |
| SMOTE | **45.7±0.1** |

**Table 2.5:** Average number of "real" features over two trials.

### 2.4.2 Inter/Extra-polated Network Flows

To generate customized samples, or more precisely, inter-/extrapolating existing samples, we employ the technique described in Section 2.3.5. We then evaluate it using the success rate, which represents how accurate the control is (in % relative to all generated samples). The control quality for each feature, level, and transport layer flag is shown in Table 2.6, Table 2.7 and Table 2.8 respectively. When it comes to features, we notice that not all of them are easily controllable. For example, the total number of packets and the maximum IAT can be controlled with high precision, but the minimum IAT is difficult to control. TCP flags, on the other hand, achieve nearly equal scores with the exception of the RST flag. After a thorough investigation it became clear that a low score is induced by very few training samples with RST set, resulting in a poor learning. Finally, we show in Table 2.7 the success rate per level for the dynamic conditioning. All levels achieve roughly comparable scores, with the exception of the mid level, which has a slightly lower score. We believe that the reason is that the mid level condition has two constraints: the predicted value should be lower than $Q_{66}$ **and** higher than $Q_{33}$, whereas the other levels have only one constraint, i.e., either larger than $Q_{66}$ or smaller than $Q_{66}$ which reflects better flexibility.

### 2.4.3 From FlowGan to PacketGan

Now we have investigated the *FlowGan* architecture and its remarkable performance, we propose a further improvement to generate complete sequences of packets rather than flow features. In Figure 2.3, we propose *PacketGan*, another GAN-based concept that can, using a class condition, generate a port scan flow, for example, or a HTTP exchange depending on the training data. The core components are the generator and the (recurrent) discriminator, but because we are generating a sequence of packets that supposedly belong to the same flows, we need to keep track of previous knowledge (previous packets). To accomplish this, the recurrent generator and discriminator are both LSTM networks.

| **Feature** | **Success Rate** |
|---|---|
| Flow Bytes/s | 59.3% |
| Flow Duration | 85.5% |
| Flow IAT Max | 92.0% |
| Flow IAT Min | 54.2% |
| Packet Length Mean | 91.1% |
| Total Backward Packets | 84.4% |
| Total Fwd Packets | 96.5% |

**Table 2.6:** Success rate by feature.

| **Level** | **Success Rate** |
|---|---|
| Low | 84.9% |
| Mid | 74.2% |
| High | 80.8% |
| Overall | 80.0% |

**Table 2.7:** Success rate by level.

| **Feature** | **Success Rate** |
|---|---|
| ACK Flags | 79.5% |
| PSH Flags | 87.2% |
| RST Flags | 50.0% |
| SYN Flags | 73.7% |

**Table 2.8:** Success rate for binary features.

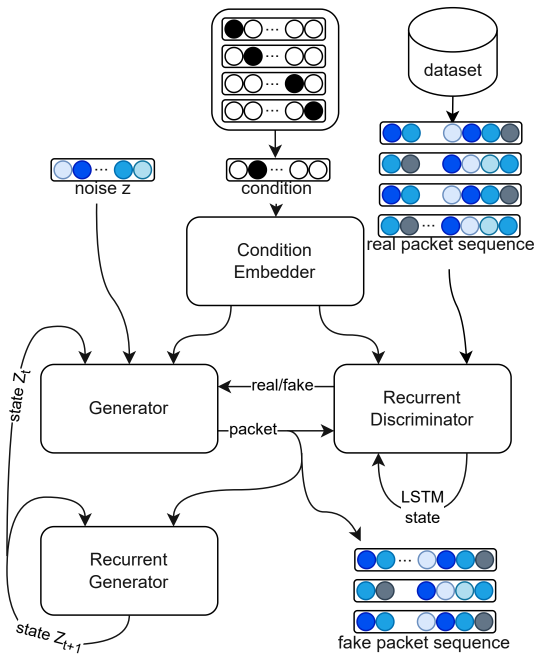


**Figure 2.3:** *PacketGan*, an architecture proposed to generate infinite sequences of packets.

To understand how the proposal works, we clarify each step in the following accordingly. At each iteration, the generator is supposed to generate a single packet that belongs to the same flow, so it takes 1) an input condition, which in this case is passed through an embedding layer simply to create a better latent space. 2) a noise vector, and 3) a state vector that represents the previous sequence of packets and is used as a helper by the generator to know which packet to generate next. The generator then generates a packet and sends it to both the recurrent discriminator and recurrent generator. Given a GT with real sequences and the previous sequence of generated packets, the task at the recurrent discriminator level is to predict whether the current packet is fake or not. This is accomplished by utilizing the LSTM network's internal capabilities to retain old knowledge and use it in combination with the current generated packet, we then obtain a label representing whether the packet is fake or real. On the other hand, the recurrent generator's task is to take the previous state, the fake packet, and return the next state that the generator uses to generate the next packet. The preceding procedure can be repeated indefinitely, resulting in an infinite sequence of packets.

Training is performed similarly to *FlowGan*, but with the following considerations: 1) The generator and recurrent generator weights are optimized using the loss resulting from the recurrent discriminator's label (therefore trying to improve the generation quality). 2) The loss obtained by combining real sequences from a dataset and fake sequences generated by the generator is used to train the recurrent discriminator. Both losses are averaged over the generated sequence of packets. It is also worth noting that in this scenario, a packet is represented as a vector of packet header features (e.g., length, direction, time to next packet, transport flags etc.). A preliminary implementation with CGAN-GP is also publicly available[2].

## 2.5 Conclusion

In this chapter, we use GANs to once more improve ML-based classification schemes for network traffic and solve the problem of network dataset augmentation and balancing. We apply four well-known GAN architectures: CGAN, WGAN, AC-GAN, and AC-WGAN, to the CICIDS2017 dataset and explore two control mechanisms for attack flow generation. The first approach uses a binary condition vector to control the generation process and allows for data balancing with only limited control (on the class only). The second approach aims for more fine-grained control over the generation process by conditioning the values of a set of eleven features therefore, allowing for a better tuning. Overall, our approach achieves high performance regarding quality of generated samples compared to a state-of-the-art-method across three tests, i.e., t-test, distance measure and detection accuracy with an external classifier; indicating that the proposed framework can be used to balance minority classes in network traffic datasets or generally for augmenting similar datasets.

[2]`github.com/CN-TU/packetGan`

When compared to the state of the art, our technique produces better results in terms of representativeness of generated flow samples and similarity across various criteria to real samples. Future work includes using the proposed *PacketGan* architecture for packet sequence generation in which the framework is supposed to construct the entire sequence of packets of a given flow instead of aggregate statistics.

CHAPTER 3

# Optimized Neural Networks for Faster Detection

***Notice of adoption from previous publications - 8***
*Parts of this section's content have been published in the following paper:*
*Fares Meghdouri, Maximilian Bachl, and Tanja Zseby. Eagernet: Early predictions of neural networks for computationally efficient intrusion detection. In* 2020 4th Cyber Security in Networking Conference (CSNet)*, pages 1–7. IEEE, 2020*[a]

[a]The main author proposed the idea and performed the evaluation whereas the second co-author helped with the implementation.

Fully Connected Neural Networks (FCNNs) have gained remarkable attention in recent years due to the availability of data and computational resources for research. Many state-of-the-art architectures currently used in different applications and especially in network traffic analysis are effectively based on simple FCNN architectures.

## 3.1 Introduction

Since the optimal size of a neural network cannot be mathematically derived, experts use a combination of rule-of-thumb and fine tuning methods to define the best architecture that fits their needs. Yet, some experiments show that the more layers (and neurons) a network contains, the better its learning performance becomes [76].

Elaborating on what is presented in Part I/Chapter 3, a typical FCNN might run into two inconveniences if it's extremely deep: 1) the loss during training needs more time to converge and 2) the time necessary to make a prediction (forward-pass) is proportional to the number of layers. Even though most modern hardware is capable of performing a forward-pass without significant delay, prediction speed is still relevant in time critical applications such as network intrusion detection.

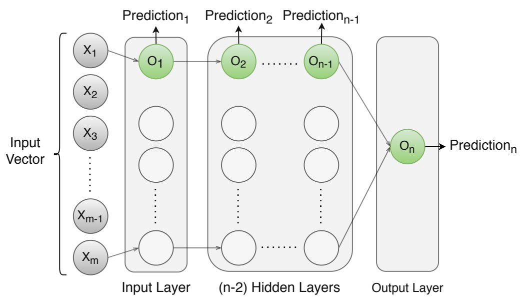
To obtain early and fast predictions, we propose a new architecture that, on the one hand, uses the entire network capacity for learning network attack classification and, on the other hand, exits the forward-pass and yields predictions as soon as the confidence reaches a certain threshold. In other words, it makes it possible to avoid evaluating the entire network for some samples. We call different FCNNs architectures that use our proposed approach *EagerNets* (Eager Stopping Networks). EagerNets therefore allow, where possible, the reduction of computing resources and energy usage while achieving similar detection performance to that of a full forward-pass.

## 3.2 Proposal: EagerNet

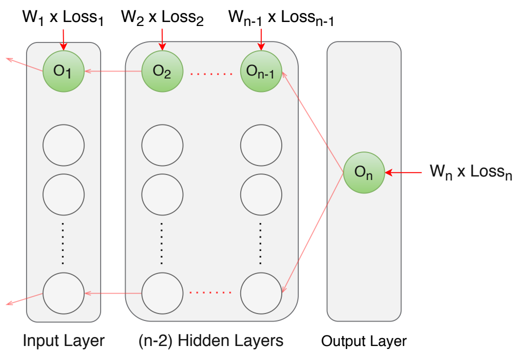
### 3.2.1 Concept

To begin with, we extract flow samples using the CAIA representation (see Part I/Chapter 4) for all our experiments and set an observation timeout of 1,800 seconds, after which we terminate flows. Each flow is considered to be one data sample and has two sets of labels: a binary (attack/benign) label and also a categorical label which represents the attack family (which is then one-hot-encoded). Our initial objective is to train a classifier until it can correctly classify unseen flows and can also determine what kind of attack class they belong to.

Because some network attack families are simpler to detect compared to others we modify the standard FCNN architecture so that, for easily classifiable samples, the network doesn't have to evaluate all layers. Our novel architecture is based on the assumption that deep neural networks can learn increasingly complex functions at each subsequent layer which in turn are only required for the classification of some particular notoriously hard-to-classify samples. As a result, we build a network such that it contains an additional output neuron at each layer, allowing for early predictions. The proposed *EagerNet* architecture is shown in Figure 3.1a. Initial neurons are shown in gray and output neurons (per layer) are shown in green ($O_i$). When a network flow is observed and passed through the entire preprocessing chain, the resulting data sample $\mathbf{x}$ is passed to the network via the input layer. The neural network proceeds evaluating the layers one by one and yields a confidence value at each layer. Here we define the confidence as a value representing how certain the network is about a prediction. It is a scalar between 0.5 and 1 obtained by applying the sigmoid function [1] to the output neuron and then computing the distance to the farthest label (e.g. if the scalar is 0.2, the farthest label is 1 we obtain a confidence of 0.8 (or 80%) that the sample belongs to class 0). By looking at the confidence the neural network determines whether additional layers should be evaluated or if the currently obtained confidence is high enough. This methodology ensures that simpler samples are classified early in the forward-pass and more complex patterns are passed into deeper layers. In Section 3.4.3, we show that on average *EagerNet* reduces resource consumption, while achieving comparable performance to normal FCNN.



**(a)** EagerNet: Forward-pass.



**(b)** EagerNet: Back-propagation of gradients.

**Figure 3.1:** The difference between conventional neural networks and our proposed architecture. *EagerNet* has additional output neurons at each layer (in green).

### 3.2.2 Network Architecture

For our experiments, we use a neural network that consists of as many layers as needed to achieve comparable accuracy with recent work. Leaky ReLU activations [161] ($\alpha = 0.1$) are applied after each layer and Adam is used as loss optimizer with $lr = 0.001$. To improve training, we use neurons dropout ($r = 0.2$). The loss used function is defined in what follows.

### 3.2.3 Binary vs. Multiclass Classification

We experiment with two different variants. The first one consists of a binary classifier (i.e. a 1 is an attack flow and a 0 is a benign flow). The predictions in this case are given by a single output neuron followed by a sigmoid activation function to obtain probabilities. During back-propagation, we use a binary cross-entropy loss and average the loss over the batch. Equation (3.2.1). shows how the loss is determined for a batch of size $N$, with the $n^{th}$ input being $x_n$, a class $y_n$ and the sigmoid activation function $\sigma(\cdot)$. The loss term is actually constructed by combining multiple losses from each layer as shown in Figure 3.1b. The losses are combined into a single loss using weighting as explained in Table 3.2.

$$l = -\frac{1}{N}\sum_{n}^{N}[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n))] \qquad (3.2.1)$$

For the multiclass approach, on the other side, there is one output neuron per attack family thus, each neuron indicates how likely it is, that the currently processed sample belongs to the respective class. During back-propagation, we use the categorical cross-entropy loss to penalize all outputs together, so there can be no overlap between classes. Equation (3.2.2). shows how the loss is determined for a batch of size $N$, with the $n^{th}$ input $x_n$ and the softmax activation function [2] $\phi(.)$ (to allow for only one active class), $C$ being the number of classes or output neurons and $i$ is the current class/output. Similarly to before, the loss term is derived from combining losses at all layers.

$$l = -\frac{1}{N}\sum_{n}^{N}\sum_{i}^{C} y_{n,i} \cdot \log(\phi(x_n)_i) \qquad (3.2.2)$$

While traditional FCNNs use one loss term for optimization, we show here how our proposal uses the losses from all layers which allows for learning trivial mapping functions earlier in the network.

### 3.2.4 Combining the Loss

Conventional FNN uses the loss of the last layer to adjust the gradients. In this proposal, however, we have multiple outputs; one at each layer so we have to aggregate losses and optimize based on a single term. Figure 3.1b demonstrates how losses are used for back-propagation from the respective layer only. As a consequence, the weights of the last layer are affected only by the loss of the last output. Similarly, the weights of the first layer are affected by the losses of the entire network. In addition, we introduce a weighting policy for losses. We allocate a weight to each loss that is predefined prior to training. The aim is to assist the network determine which layer is more important. Three types of weights are defined:

- Uniform weights. all losses have the same impact:

$$\mathcal{W}_{\mathcal{U}} = \{W_0, ..., W_n\} = \{\frac{1}{\sum_{i=0}^{n} 1}, ..., \frac{1}{\sum_{i=0}^{n} 1}\} \qquad (3.2.3)$$

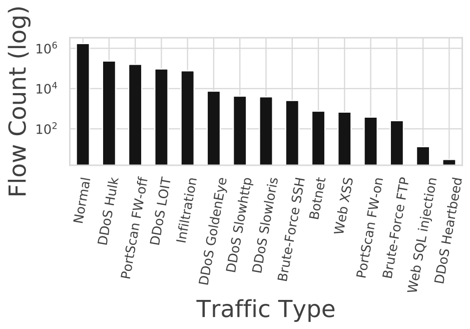
- Increasing weights. losses are increasingly important from first to last layer.

$$\mathcal{W}_{\mathcal{I}} = \{W_0, ..., W_n\} = \{\frac{1}{\sum_{i=0}^{n} i + 1}, ..., \frac{n+1}{\sum_{i=0}^{n} i + 1}\} \qquad (3.2.4)$$

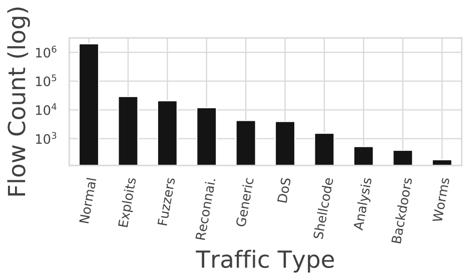
- Decreasing weights. losses are decreasingly important from first to last layer.

$$\mathcal{W}_{\mathcal{D}} = \{W_0, ..., W_n\} = \{\frac{n+1}{\sum_{i=0}^{n} i + 1}, ..., \frac{1}{\sum_{i=0}^{n} i + 1}\} \qquad (3.2.5)$$

The effect of weight distributions is discussed in Section 3.3.



(a) CICIDS2017



(b) UNSW-NB15

**Figure 3.2:** Traffic distribution represented by number of flows.

## 3.3 Evaluation

In what follows we discuss obtained results and observations. Overall, *EagerNet* shows promising performance that are comparable to FCNNs which in the contrary always use the entire network thus, wasting resources to some extent. Interested readers can use models and reproduce experiments with the publicly available code base[1].

For our experiments, we use the two intrusion detection datasets discussed throughout this work: CICIDS2017 [221] and UNSW-NB15 [177]. We extract flow samples utilizing the CAIA feature vector yielding 2,317,922 and 2,065,224 flows, respectively, from CICIDS2017 and UNSW-NB15 (see Figure 3.2). We then eliminate duplicate instances and standardize the features using z-score. We use a split of $\frac{2}{3}$ for training and $\frac{1}{3}$ for evaluation and testing.

## 3.4 Result and Discussion

After training, we use the weights obtained at the 800th epoch of each network for evaluation. We aggregate results obtained from using different weighting strategies and the basic FCNN and present them in the following sections.

### 3.4.1 Comparability to FCNNs

In Table 3.1, we show the F1 score obtained at the last output layer of two neural networks: a bare FCNN and *EagerNet* with the same number of layers and neurons. Here we use the F1 score as a metric capable to handle class imbalance. Results suggest that on average, *EagerNet* achieves similar scores to a conventional FCNN if used as one. We note that in this experiment, *EagerNet* is not used as intended because predictions are taken only from the last layer hence, ignoring the major benefit of our proposal. Using

[1] `github.com/CN-TU/ids-backdoor/tree/eager`

the confidence level to exit the forward-pass whenever possible might lead to higher accuracy if middle layers are optimized better in predicting certain classes/patterns.

| **Dataset** | **Variant*** | **FCNN** | **EagerNet** |
|---|---|---|---|
| CICIDS2017 | Binary | 0.989 | 0.993 |
| | Multiclass | 0.979 | 0.920 |
| UNSW-NB15 | Binary | 0.919 | 0.908 |
| | Multiclass | 0.882 | 0.880 |

* All architectures consist of 10 layers × 64 neurons in addition to input and output layers.

**Table 3.1:** F1 scores at the last layer of each architecture.

### 3.4.2 Weighted Loss

In Table 3.2 we gather results from extensive training by building different networks each with a different number of hidden layers and weight distribution but all with 128 neurons per layer. The table shows four metrics accuracy, precision, recall and Youden's J score obtained at the last layer (by setting the confidence threshold to the maximum). In the CICIDS2017 dataset case, scores show that the weight distribution and the architecture itself (number of layers) have little effect on performance. Since accuracy didn't improve when increasing the number of layers, this means that most instances are are evaluated with high confidence after the few first layers which makes the classification task for CICIDS2017 manageable even for shallow neural networks. Results of UNSW-NB15 show a slightly different behavior. Uniformly weighting losses gives the best scores on average and, in addition, deeper architectures outperform the shallower ones. This implies that network patterns in this specific dataset are more complex than in the previous one, requiring more non-linearities to learn the optimal *input* → *prediction* mapping function.

### 3.4.3 Accuracy-Speed Tradeoff

Figure 3.3 shows the accuracy-resources balance in terms of confidence threshold. Four different networks for both datasets are evaluated. We train two networks: 128 neurons by 3 layers and 64 neurons by 12 layers. The confidence threshold is shown on the horizontal axis, the average number of layers used over all samples and the average accuracy achieved are both shown on the vertical axis. Curves show promising and consistent results, whereby the network always needs a few layers for the majority of samples and only continues to evaluate further layers if an extremely high confidence level is required ($> 0.95$). We further observe two phenomena:

- The accuracy obtained with the CICIDS2017 dataset increases almost linearly in parts with respect to confidence and reaches its maximum when all layers are evaluated.

| Variant | Weights | Layers | Acc. | Prec. | Rec. | J. |
|---|---|---|---|---|---|---|
| CICIDS2017 | Uniform | 5 | 0.996 | 0.996 | 0.990 | 0.989 |
| | | 3 | 0.997 | 0.997 | 0.990 | 0.990 |
| | Increasing | 5 | 0.997 | 0.998 | 0.990 | 0.989 |
| | | 3 | 0.997 | 0.997 | 0.990 | 0.989 |
| | Decreasing | 5 | 0.997 | 0.998 | 0.989 | 0.989 |
| | | 3 | 0.997 | 0.997 | 0.990 | 0.990 |
| UNSW-NB15 | Uniform | 5 | 0.988 | 0.828 | 0.867 | 0.860 |
| | | 3 | 0.988 | 0.856 | 0.800 | 0.795 |
| | Increasing | 5 | 0.988 | 0.853 | 0.821 | 0.816 |
| | | 3 | 0.988 | 0.844 | 0.817 | 0.811 |
| | Decreasing | 5 | 0.988 | 0.843 | 0.824 | 0.819 |
| | | 3 | 0.988 | 0.851 | 0.819 | 0.813 |

Note: All neural networks consist of 128 neurons per layer.

**Table 3.2:** Effect of weights on the performance of different networks

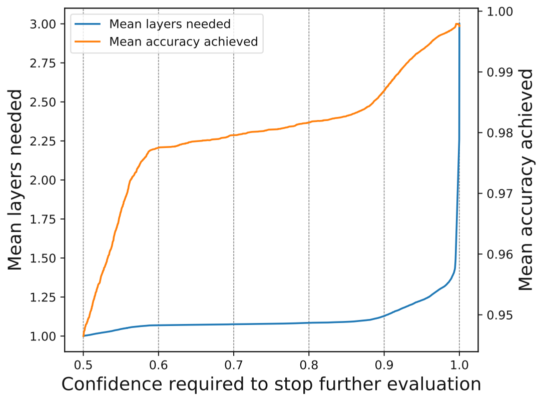
- The accuracy obtained with the UNSW-NB15 is almost unchanged when the confidence level is increased from ≈ 0.62 until ≈ 0.98 where it leaps noticeably when an additional layer is evaluated. This suggests that some patterns were "learned" only after a specific layer and hence, such samples were suddenly correctly classified.

Overall, *EagerNet* confirms that it is possible to trade a tiny percentage of accuracy in order to save a significant amount of computation resources and inference time especially in very deep neural networks.

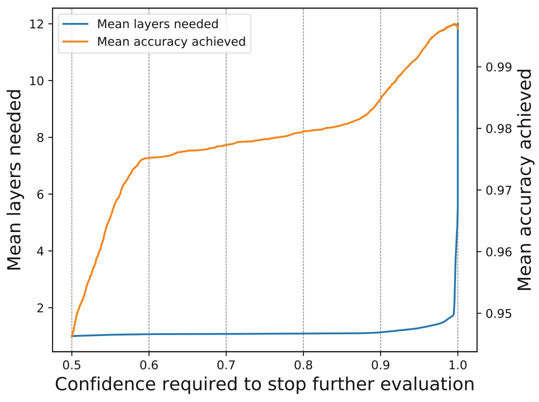
**Optimal Confidence Threshold** During deployment, the optimum confidence threshold is set by the user and plays a role in the overall achieved accuracy and the saved resources. Setting the threshold too low can cause the network to stop the forward-pass at an early stage, thereby using fewer resources but achieving low accuracy. Similarly, setting the threshold too high can cause the network to always use all layers wasting valuable resources sometimes. The desired confidence level can be therefore determined after the training phase using a similar accuracy graph as shown in Figure 3.3.

### 3.4.4 What Did the Network Learn?

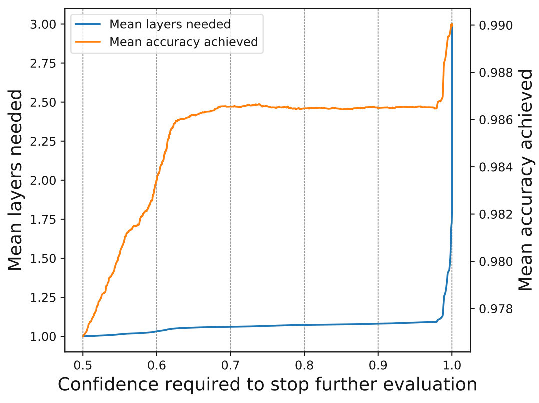
In order to understand how *EagerNet* determines at which layer the forward-pass shall be stopped, we use the multiclass classification variant and conduct an explanatory analysis. The reason why we use the multiclass variant is to observe, per attack, how many layers are on average needed. In Figure 3.4, we show the accuracy obtained by four different networks per layer and attack family. Results reflect two conclusions:



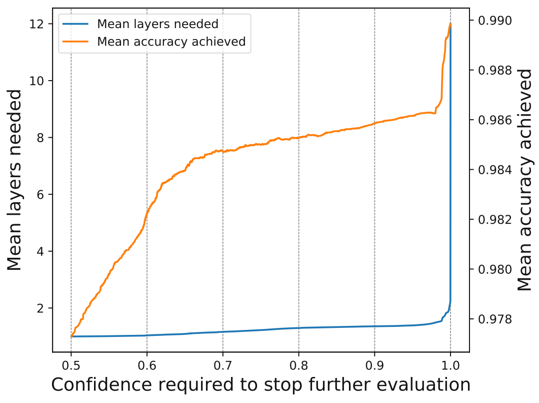
**(a)** CICIDS2017 (3 layers × 128 neurons).



**(b)** CICIDS2017 (12 layers × 64 neurons).



**(c)** UNSW-NB15 (3 layers × 128 neurons).

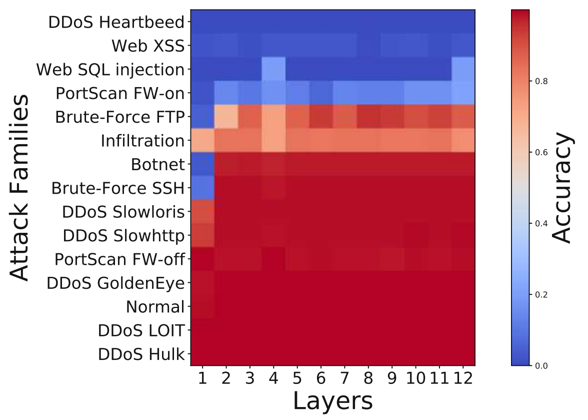


**(d)** UNSW-NB15 (12 layers × 64 neurons).

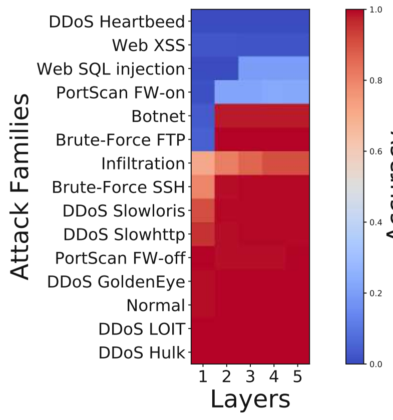
**Figure 3.3:** Confidence threshold effect on accuracy and number of needed layers (or saved resources). The maximum achieved accuracy does not represent the score obtained at the last layer but with a combination of best layers depending on the selection. Therefore, the last layer's accuracy alone might be lower.

- On average, the accuracy increases the deeper we get into the neural network. This is somewhat expected, as more layers allow for more abstraction of the input space and thus better separation of instances. However, only few layers are needed to obtain the maximum accuracy per attack family. This implies that the *EagerNet* approach significantly decreases resources when used for network intrusion detection yet keeping the detection accuracy high.

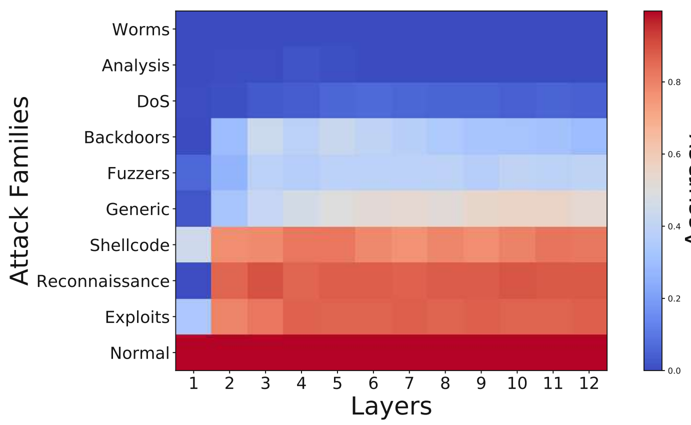
- On the other hand, some attack families (Analysis, Worms, Web XSS, DDoS Heart-bleed etc.) have extremely poor accuracy. This is because too few samples are used/available for training therefore, the neural network struggle to learn and optimize the loss for such "fragile" information because the its loss is insignificant compared to samples with higher occurrence.



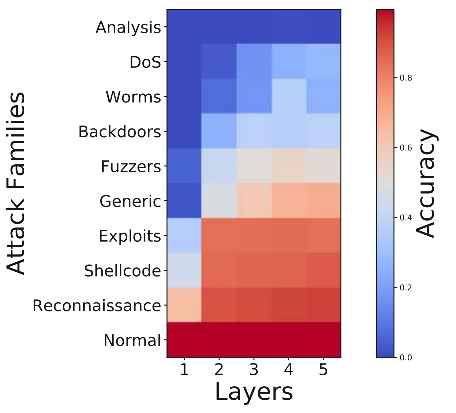
**(a)** CICIDS2017 (12 layers × 64 neurons).



**(b)** CICIDS2017 (5 layers × 128 neurons).



**(c)** UNSW-NB15 (12 layers × 64 neurons).



**(d)** UNSW-NB15 (5 layers × 128 neurons).

**Figure 3.4:** Accuracy achieved on a test set per layer and attack family (sorted vertically). Some attacks have poor accuracy because only few samples are available (refer to Figure 3.2). The effect of over-sizing the neural network immediately appears confirming the advantage of *EagerNet* whereby some attacks (e.g., DDoS) need only a few layers whereas for others (e.g., FTP Brute-Force) more layers are required to achieve the same accuracy.

### 3.4.5 Separate Or Combined Optimization of Losses?

Another option is to optimize each layer separately, in which only the loss obtained at each layer is used for tuning and back-propagation. This can be used during training to save computational resources by optimizing based on a single loss rather than the accumulation of losses up to the current layer. Experiments in this setting show that this variant performs worse possibly because a complete optimization of the network's weights is necessary. Back-propagation with combined losses is therefore required for complete and balanced training of the *EagerNet* architecture.

## 3.5 Conclusion

In this chapter, we propose an architecture for improving the classification chain. We investigate the feasibility of speeding up a ML-based classifier by terminating the evaluation of intermediate layers of a FCNNs earlier and saving computational resources. To that end, we propose *EagerNet*, an architecture that allows predictions to be made as soon as the neural network is sufficiently confident, saving energy, resources, and time; allowing similar architectures to be implemented in time-critical applications (e.g., IDSs) where detection speed is relevant. We test our approach with two intrusion detection datasets and obtain results that are comparable to conventional FCNNs while only using parts of the neural network for inference. The results support the argument that many network traffic flows are easily classifiable using a few layers and only a small percentage requires further evaluation through the network to extract more complex patterns. Furthermore, we demonstrate that by specifying a confidence level, it is possible to trade detection accuracy for resource usage, speed, and energy consumption. Consequently, the simplicity and effectiveness of the *EagerNet* architecture make it an ideal lightweight solution for potential FCNN-based IDSs when resources are of primary concern.

# Summary

In Part III, I introduced three approaches to enhance the state-of-the-art in ML-based traffic classification and anomaly detection. In Chapter 1, I propose ODM, a sampling algorithm for extracting data coresets that provides a low-density model of a given dataset. Coresets aid various algorithms in training, stability, and generalization. In Chapter 2, I address the issue of data imbalance in network traffic, in which some classes are rarely encountered. To that end, I present a framework for data balancing and controllable network traffic generation at both,flow and packet levels. In Chapter 3, I tune the vanilla FCNNs training process to allow for faster inference while also using less resources. These solutions are applied to various parts of the typical ML-based NTA pipeline, but the objective is always the same: enhancing training and performance accuracy.

The evaluation phase consistently shows that classification/detection accuracy improves. Coresets, for example, help to remove noise from data and speed up training by using fewer samples keeping the same structure as the original dataset. Data augmentation using GANs allows balancing network datasets with class imbalance which otherwise cause problems when employing label-distribution-sensitive algorithms. The evaluation confirms that generated samples are similar to some extent to those recorded. In the same line, I present an extension to augment the data with network packets instead of network flows. Finally, tailored training of DNNs by minimizing the loss at each layer is shown to contribute in generating quicker predictions. It also confirms that not all layers must be evaluated for accurate anomaly detection, saving systems time and resources.

By combining proposals in this and the previous part (Part II), my effort seeks to improve ML for network traffic analysis as a whole.

# Part IV

# Conclusion

## 1.1 Findings

In this thesis I present a comprehensive set of techniques and approaches for enhancing NTA and anomaly detection with ML. The proposals are divided into two main parts, each consisting of several chapters that deal with different aspects.

In the first part, the focus is on feature engineering, which is shown to be crucial for the success of subsequent operations. The proposed representations include three network traffic representations for solving different anomaly detection tasks, a novel approach for aggregating network flows across multiple layers (*multi-key*), a method for achieving similar performance without flow aggregation (*I-Notice*), and a minimal feature vector for counting communication flows in VPN tunnels (*FlowCount*). The final chapter investigates the impact of data scaling on the input space.

In the second part, three approaches are proposed for enhancing the state-of-the-art in ML-based traffic classification and anomaly detection. The first approach involves the development of an algorithm for extracting data coresets (*ODM*), which provide a low-density model of a given dataset and aid various algorithms in training, stability, and generalization. The second approach addresses the issue of data imbalance in network traffic by presenting a framework for data balancing and controllable network traffic generation at both the flow and packet levels (*Flow/PacketGAN*). The third approach involves tuning the vanilla FCNN training process to allow for faster inference while also using fewer resources (*EagerNet*).

In conclusion, the proposed solutions aim to enhance the existing ML approaches for NTA holistically. The evaluation phase consistently show that the performance metrics have improved, indicating that the research questions presented in Part I have been successfully addressed. These findings provide promising opportunities for future research in the field of NTA.

In the subsequent sections, I outline potential research inquiries that merit exploration but remain outside the scope of this thesis. Additionally, I suggest potential directions for future research, delineating possibilities for enhancing the state of the art.

## 1.2 Open Questions

Several questions within the thesis remain open and require further investigation. This section highlights a set of open questions that require additional research, as well as those that are subject to subjective interpretation, they are summarized as follows:

- What are the limitations of using Supervised Learning for network anomaly detection, particularly in detecting zero-day attacks where patterns are not known a-priori? While Supervised Learning-based methods have shown promising performance in evaluation benchmarks, what are the potential failure cases and can these be addressed through the integration of Unsupervised Learning?

- Throughout this work, I addressed traffic encryption and limitations imposed to most analysis methods causing performance degradation. The questions is thus, whether encrypted header features can be inferred using available data post-encryption, and would generative models be effective in this regard? One possible approach is to predict the application or protocol number based on the sequence of packet lengths, given that applications exhibit similar patterns due to the underlying data generation mechanism. While this question was not addressed in this thesis, exploring this direction could potentially alleviate the challenges associated with encryption and its impact on analysis methods.
- What are the implications of encryption using QUIC for traffic analysis, and how do they differ from well established encryption protocols such as TLS and IPsec? Given that QUIC employs a unique method of aggregating connections via the "connection ID", what other analytical approaches are necessary to handle such traffic compared to others?
- How can the generalization of conclusions about data scaling, transformation, and preprocessing be ensured beyond the specific datasets examined in Part II - Chapter 4? While the four datasets used in the respective study provide some insights, it is uncertain whether the observed patterns and guidelines hold true for other datasets or timeframes, given the constantly evolving nature of network traffic.
- How can the transferability of trained ML models across diverse environments be established, given that models trained on specific datasets may perform poorly when tested on different setups? While Part II (Chapter 2 and Chapter 3) of this thesis partially evaluates the performance of extracted models on other datasets, the broader question of whether these models perform comparably well on new or unseen data remains open. This is an important aspect as network data constantly emerges and operators have to ensure the model's continued applicability and effectiveness.
- Finally, to what extent can the proposed methods and solutions presented in this thesis be effectively applied to practical applications? What changes or adaptations are necessary to enable successful deployment in real-world scenarios? Moreover, what are the next critical steps required to evaluate the performance of all proposals in real networks, given that this was not possible during the course of this research?

## 1.3 Future Work

The proposed methodologies offer ample opportunities for extension, enhancement, and adaptation to other applications or scenarios based on the open questions raised previously. While future work is elaborated at the conclusion of each proposal, the following summary highlights potential directions for further investigation and improvement that are deemed noteworthy.

- One potential future research in the context of the *multi-key* feature vector is conducting feature selection to reduce the number of features and preserve solely the relevant ones. This includes investigating the reason why some features are more important and exploring whether their relevancy varies under different test conditions.

- A potential extension to the flow counting problem is to investigate methods to separate flows after they have been counted. Accurate flow count estimation can be used as an input parameter for such experiments. Successfully addressing the challenge of flow separation and labeling would present opportunities for significant improvements, particularly for network monitoring systems, especially in the case of harsh encryption schemes such as IPsec. A first attempt to solve this issue has been presented in [105].

- The data processing experiments conducted in Part II - Chapter 4, apart from the applications of anomaly detection and NTA, have the potential to be extended to other transformation techniques and domains. Future work should include a more comprehensive analysis of various aspects, such as applicability to UL, RL, etc., that were not explored in this study. Preprocessing data is a complex and challenging task due to the various uncertainties in the data, and therefore, it is necessary to approach it with caution. Hence, the analysis presented in this study was limited to a few manageable methods.

- Given the success of the ODM approach in extracting low-density models from data, it is reasonable to investigate its potential in deriving feature relevancy based on entropy. Specifically, the algorithm's internal mechanism involves moving observers in multiple directions, so by tracking the gradient of each observer, one can infer which dimension has the most variance and hence might reflect a high relevancy. Since higher entropy implies more information, the observer's movement may provide a good indication of more discriminant features.

- Future work in connection with ODM also includes developing a variant of the algorithm that deals with stream data in such a way that it does not store all previous observers but rather only the relevant ones. This adaptation enables deployment in environments where data is constantly changing, such as communication networks, in which clusters appear and fade over time. This change enables ODM to extract dynamic coresets for better data representation.

- The proposal presented as *FlowGan* has potential for extension to generate a wider range of attributes beyond those currently specified. The framework's versatility and flexibility enable its extensibility. With this approach, *FlowGan* can be tailored to each of the feature vectors examined in this thesis, providing opportunities for generating diverse types of network traffic.

- In addition to the *FlowGan* proposal, further development of the second proposal, *PacketGan*, may focus primarily on enhancing the evaluation process. Currently, assessing the legitimacy of generated packets is a complex task that entails multiple evaluation stages. As part of future work, the evaluation benchmark can be refined to effectively assess the quality of trained models for realistic packet generation.

- As noted in the open questions section, the continuous evolution of QUIC encrypted traffic requires that current analysis frameworks be adapted accordingly. The proposed methods in this thesis also require adjustment to account for QUIC traffic while preserving comparable performance. Notably, numerous research papers on this subject have been published recently, indicating a need for further advancement in this area.

# List of Figures

# List of Tables

# Acronyms

**1D-CNN** One-Dimensional Convolutional Neural Network.

**AC-WGAN** Wasserstein GAN with Auxiliary Classifier.

**AC-GAN** Auxiliary-Classifier GAN.

**AE** Auto-Encoder.

**AI** Artificial Intelligence.

**ALE** Accumulated Local Effect.

**BCE** Binary Cross Entropy.

**CDD** Critical Difference Diagram.

**CGAN** Conditional GAN.

**CNN** Convolutional Neural Network.

**CNN** Condensed Nearest Neighbour.

**CV** Computer Vision.

**DDoS** Distributed Denial of Service.

**DL** Deep Learning.

**DNN** Deep Neural Network.

**DNS** Domain Name System.

**DOS** Denial of Service.

**DPI** Deep Packet Inspection.

**DT** Decision Tree.

**DTLS** Datagram Transport Layer Security.

**ESP** Encapsulating Security Payload.

**FCNN** Fully Connected Neural Network.

**FIFO** First-In First-Out.

**FNN** Feed-forward Neural Network.

**FPR** False Positive Rate.

**GAN** Generative Adversarial Network.

**GB** Gradient Boosting.

**GDPR** General Data Protection Regulation.

**GNB** Gaussian Naive Bayes.

**GNG** Growing Neural Gas.

**GRU** Gated Recurrent Network.

**GT** Ground Truth.

**HTTP** Hypertext Transfer Protocol.

**HTTPS** Hypertext Transfer Protocol Secure.

**IANA** Internet Assigned Numbers Authority.

**IAT** Inter-Arrival Time.

**ICMP** Internet Control Message Protocol.

**IDS** Intrusion Detection System.

**IKE** Internet Key Exchange.

**I-Notice** Incremental Convolutional Classifier for Network Packets.

**IoT** Internet of Thing.

**IP** Internet Protocol.

**IPFIX** IP Flow Information Export.

**IPsec** IP Security.

**KMC** k-Means Coreset.

**k-Means** k-Means.

**k-NN** k-Nearest Neighbors.

**LIME** Local Interpretable Model-agnostic Explanations.

**LSTM** Long Short Term Memory.

**MAE** Mean Absolute Error.

**MAPE** Mean Absolute Percentage Error.

**MAWI** Measurement and Analysis on the WIDE Internet.

**MC** Markov Chain.

**ML** Machine Learning.

**MLP** Multilayer Perceptron.

**MSE** Mean Squared Error.

**NB** Naive Bayes.

**NID** Network Intrusion Detection.

**NIDS** Network Intrusion Detection System.

**NLP** Natural Language Processing.

**NTA** Network Traffic Analysis.

**ODM** Observer-based Data Modeling.

**OHE** One-Hot-Encoding.

**OS** Operating System.

**PCA** Principal Component Analysis.

**PCAP** Packet CAPture.

**PDP** Partial Dependence Plot.

**QoS** Quality of Service.

**QUIC** Quick UDP Internet Connection.

**RF** Random Forest.

**RL** Reinforcement Learning.

| | |
|---|---|
| **RNN** | Recurrent Neural Network. |
| **RS** | Random Sampling. |
| **SA** | Security Association. |
| **SDO** | Sparse Data Observer. |
| **SHAP** | SHapley Additive exPlanations. |
| **SL** | Supervised Learning. |
| **SMOTE** | Synthetic Minority Oversampling Technique. |
| **SOM** | Self-Organizing Map. |
| **SPAN** | Switch Port ANalyzer. |
| **SSH** | Secure SHell. |
| **SVM** | Support Vector Machine. |
| **TAP** | Test Access Point. |
| **TCP** | Transmission Control Protocol. |
| **TLS** | Transport Layer Security. |
| **TPR** | True Positive Rate. |
| **TTL** | Time-to-Live. |
| **UDP** | User Datagram Protocol. |
| **UL** | Unsupervised Learning. |
| **VPN** | Virtual Private Network. |
| **WGAN** | Wasserstein GAN. |
| **WGAN-GP** | Wasserstein GAN with Gradient Penalty. |
| **WGM** | Weighted Gaussian Mixture. |
| **XAI** | eXplainable Artificial Intelligence. |

# Bibliography

[1] Sigmoid function, June 2020. Page Version ID: 964386757.

[2] Softmax function, July 2020. Page Version ID: 968517329.

[3] Naoki Abe, Bianca Zadrozny, and John Langford. Outlier detection by active learning. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 504–509. ACM, 2006.

[4] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. Mobile encrypted traffic classification using deep learning. In *2018 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE, 2018.

[5] Oluwamayowa Ade Adeleke, Nicholas Bastin, and Deniz Gurkan. Network traffic generation: A survey and methodology. *ACM Computing Surveys (CSUR)*, 55(2):1–23, 2022.

[6] Pankaj K Agarwal, Sariel Har-Peled, and Kasturi R Varadarajan. Geometric approximation via coresets. *Combinatorial and computational geometry*, 52:1–30, 2005.

[7] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. On the surprising behavior of distance metrics in high dimensional space. In *International conference on database theory*, pages 420–434. Springer, 2001.

[8] Taiwo Samuel Ajani, Agbotiname Lucky Imoize, and Aderemi A Atayero. An overview of machine learning within embedded and mobile devices–optimizations and applications. *Sensors*, 21(13):4412, 2021.

[9] Doğukan Aksu, Serpil Üstebay, Muhammed Ali Aydin, and Tülin Atmaca. Intrusion detection with comparative analysis of supervised learning techniques and fisher score feature selection algorithm. In *International Symposium on Computer and Information Sciences*. Springer, 2018.

[10] Omar Y Al-Jarrah, Paul D Yoo, Sami Muhaidat, George K Karagiannidis, and Kamal Taha. Efficient machine learning for big data: A review. *Big Data Research*, 2(3):87–93, 2015.

[11] Hashem Alaidaros, Massudi Mahmuddin, Ali Al Mazari, et al. An overview of flow-based and packet-based intrusion detection performance in high speed networks. In *Proceedings of the International Arab Conference on Information Technology*, pages 1–9, 2011.

[12] Ralph A Alexander. A note on averaging correlations. *Bulletin of the Psychonomic Society*, 28(4):335–336, 1990.

[13] Mohammad Almseidin, Maen Alzubi, Szilveszter Kovacs, and Mouhammd Alkasassbeh. Evaluation of machine learning algorithms for intrusion detection system. In *2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY)*, pages 000277–000282, September 2017. ISSN: 1949-0488.

[14] Blake Anderson and David McGrew. Identifying encrypted malware traffic with contextual flow data. In *Proceedings of the 2016 ACM workshop on artificial intelligence and security*. ACM, 2016.

[15] Fabrizio Angiulli. Fast condensed nearest neighbor rule. In *Proceedings of the 22nd international conference on Machine learning*, pages 25–32, 2005.

[16] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[17] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[18] Hagai Attias. A variational baysian framework for graphical models. In *Advances in neural information processing systems*, pages 209–215, 2000.

[19] Chen Avin, Manya Ghobadi, Chen Griner, and Stefan Schmid. On the complexity of traffic traces and implications. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2020.

[20] Olivier Bachem, Mario Lucic, and Andreas Krause. Scalable k-means clustering via lightweight coresets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1119–1127, 2018.

[21] Maximilian Bachl, Alexander Hartl, Joachim Fabini, and Tanja Zseby. Walling up backdoors in intrusion detection systems. In *Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks*, 2019.

[22] Maximilian Bachl, Fares Meghdouri, Joachim Fabini, and Tanja Zseby. Sparseids: Learning packet sampling with reinforcement learning. In *2020 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9. IEEE, 2020.

[23] Maximilian Bachl, Fares Meghdouri, Joachim Fabini, and Tanja Zseby. SparseIDS: Learning Packet Sampling with Reinforcement Learning. In *2020 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2020. arXiv: 2002.03872.

[24] Amitav Banerjee, UB Chitnis, SL Jadhav, JS Bhawalkar, and S Chaudhury. Hypothesis testing, type i and type ii errors. *Industrial psychiatry journal*, 18(2):127, 2009.

[25] Roni Bar-Yanai, Michael Langberg, David Peleg, and Liam Roditty. Realtime classification for encrypted traffic. In *International Symposium on Experimental Algorithms*. Springer, 2010.

[26] Etienne Barnard and LFA Wessels. Extrapolation and interpolation in neural network classifiers. *IEEE Control Systems Magazine*, 12(5):50–53, 1992.

[27] Elaheh Biglar Beigi, Hossein Hadian Jazi, Natalia Stakhanova, and Ali A Ghorbani. Towards effective feature selection in machine learning-based botnet detection approaches. In *2014 IEEE Conference on Communications and Network Security*. IEEE, 2014.

[28] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, pages 1798–1828, 2013.

[29] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[30] Francesco Bergadano. The problem of induction and machine learning. In *IJCAI*, 1991.

[31] Daniel S. Berman, Anna L. Buczak, Jeffrey S. Chavis, and Cherita L. Corbett. A Survey of Deep Learning Methods for Cyber Security. *Information*, 10(4):122, April 2019.

[32] Leyla Bilge and Tudor Dumitraş. Before we knew it: an empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 833–844, 2012.

[33] Jock A Blackard and Denis J Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, 24(3):131–151, 1999.

[34] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive Neural Networks for Efficient Inference. *arXiv:1702.07811 [cs, stat]*, September 2017. arXiv: 1702.07811.

[35] Ali Borji. Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019.

[36] Alessio Botta, Alberto Dainotti, and Antonio Pescapè. Multi-protocol and multi-platform traffic generation and measurement. *INFOCOM 2007 Demo Session*, 2010, 2007.

[37] George EP Box and David R Cox. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):211–243, 1964.

[38] Kendrick Boyd, Kevin H Eng, and C David Page. Area under the precision-recall curve: point estimates and confidence intervals. In *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2013.

[39] Rebekah Brown and Robert M Lee. The evolution of cyber threat intelligence (cti): 2019 sans cti survey. *SANS Institute. Available online: https://www. sans. org/white-papers/38790/(accessed on 12 July 2021)*, 2019.

[40] Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications surveys & tutorials*, 18(2):1153–1176, 2015.

[41] Anna L. Buczak and Erhan Guven. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys Tutorials*, 18(2):1153–1176, 2016.

[42] Trevor Campbell and Boyan Beronov. Sparse variational inference: Bayesian coresets from scratch. *arXiv preprint arXiv:1906.03329*, 2019.

[43] Trevor Campbell and Tamara Broderick. Bayesian coreset construction via greedy iterative geodesic ascent. *arXiv preprint arXiv:1802.01737*, 2018.

[44] Trevor Campbell and Tamara Broderick. Automated scalable bayesian inference via hilbert coresets. *The Journal of Machine Learning Research*, 20(1):551–588, 2019.

[45] Guilherme O Campos, Arthur Zimek, Jörg Sander, Ricardo JGB Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael E Houle. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery*, 30(4):891–927, 2016.

[46] Luis M Candanedo and Véronique Feldheim. Accurate occupancy detection of an office room from light, temperature, humidity and co2 measurements using statistical learning models. *Energy and Buildings*, 112:28–39, 2016.

[47] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[48] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[49] FENG Changyong, WANG Hongyue, LU Naiji, CHEN Tian, HE Hua, LU Ying, et al. Log-transformation and its implications for data analysis. *Shanghai archives of psychiatry*, 26(2):105, 2014.

[50] Jeremy Charlier, Aman Singh, Gaston Ormazabal, Radu State, and Henning Schulzrinne. Syngan: Towards generating synthetic network attacks using gans. *arXiv preprint arXiv:1908.09899*, 2019.

[51] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[52] Aiyou Chen, Yu Jin, Jin Cao, and Li Erran Li. Tracking long duration flows in network traffic. In *2010 Proceedings IEEE INFOCOM*. IEEE, 2010.

[53] Adriel Cheng. Pac-gan: Packet generation of network traffic using generative adversarial networks. In *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 0728–0734. IEEE, 2019.

[54] Md Moin Uddin Chowdhury, Frederick Hammond, Glenn Konowicz, Chunsheng Xin, Hongyi Wu, and Jiang Li. A few-shot deep learning approach for improved intrusion detection. In *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*. IEEE, 2017.

[55] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An e cient access method for similarity search in metric spaces. In *Proceedings of the 23rd VLDB conference, Athens, Greece*, pages 426–435. Citeseer, 1997.

[56] Cisco and its affiliates. White paper: Cisco encrypted traffic analytics. Technical report, Cisco, 2019.

[57] B. Claise and B. Trammell. Rfc 7012: Information model for ip flow information export (ipfix). Technical report, Internet Engineering Task Force (IETF), 2013.

[58] Benoit Claise, Brian Trammell, and Paul Aitken. Specification of the ip flow information export (ipfix) protocol for the exchange of flow information. Technical report, 2013.

[59] David Colquhoun. The reproducibility of research and the misinterpretation of p-values. *Royal society open science*, 4(12):171085, 2017.

[60] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009.

[61] Kelton AP da Costa, João P Papa, Celso O Lisboa, Roberto Munoz, and Victor Hugo C de Albuquerque. Internet of things: A survey on machine learning-based intrusion detection approaches. *Computer Networks*, 151:147–157, 2019.

[62] Manuel Gonçalves da Silva Neto and Danielo G Gomes. Network intrusion detection systems design: A machine learning approach. In *Anais do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. SBC, 2019.

[63] Alberto Dainotti, Antonio Pescape, and Kimberly C Claffy. Issues and future directions in traffic classification. *IEEE network*, 26(1):35–40, 2012.

[64] Salvatore D'Antonio, Tanja Zseby, Christian Henke, and Lorenzo Peluso. Flow selection techniques. Technical report, 2013.

[65] Mopsi data. mopsi-finland, url = https://github.com/ deric/ clustering-benchmark/ blob/ master/ src/ main/ resources/ datasets/ artificial/ mopsi-finland.arff, urldate = 2019-08-19.

[66] Jayeeta Datta, Neha Kataria, and Neminath Hubballi. Network traffic classification in encrypted environment: a case study of google hangout. In *2015 twenty first national conference on communications (NCC)*. IEEE, 2015.

[67] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006.

[68] Claudio De Stefano, Marilena Maniaci, Francesco Fontanella, and A Scotto di Freca. Reliable writer identification in medieval manuscripts through page layout features: The "avila" bible case. *Engineering Applications of Artificial Intelligence*, 72:99–110, 2018.

[69] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[70] Laura DeNardis. A history of internet security. In *The history of information security*, pages 681–704. Elsevier, 2007.

[71] Pedro M Domingos. A few useful things to know about machine learning. *Commun. acm*, 2012.

[72] Chris Donahue, Julian McAuley, and Miller Puckette. Adversarial audio synthesis. *arXiv preprint arXiv:1802.04208*, 2018.

[73] Jaimie Drozdal, Justin Weisz, Dakuo Wang, Gaurav Dass, Bingsheng Yao, Changruo Zhao, Michael Muller, Lin Ju, and Hui Su. Trust in automl: exploring information needs for establishing trust in automated machine learning systems. In *Proceedings of the 25th International Conference on Intelligent User Interfaces*, pages 297–307, 2020.

[74] Ran Dubin, Amit Dvir, Ofir Pele, and Ofer Hadar. I know what you saw last minute—encrypted http adaptive video streaming title classification. *IEEE transactions on information forensics and security*, 2017.

[75] Pablo Duboue. *The art of feature engineering: essentials for machine learning*. Cambridge University Press, 2020.

[76] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on learning theory*, pages 907–940, 2016.

[77] Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. Gansynth: Adversarial neural audio synthesis. *arXiv preprint arXiv:1902.08710*, 2019.

[78] Cristian Estan, George Varghese, and Mike Fisk. Bitmap algorithms for counting active flows on high speed links. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, 2003.

[79] Alice Este, Francesco Gringoli, and Luca Salgarelli. On the stability of the information carried by traffic flow features at the packet level. *SIGCOMM Comput. Commun. Rev.*, 2009.

[80] Alice Este, Francesco Gringoli, and Luca Salgarelli. On the stability of the information carried by traffic flow features at the packet level. *ACM SIGCOMM Computer Communication Review*, 2009.

[81] Ian W Evett and Ernest J Spiehler. Rule induction in forensic science. *KBS in Goverment*, pages 107–118, 1987.

[82] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 2006.

[83] Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for k-means, pca, and projective clustering. *SIAM Journal on Computing*, 49(3):601–657, 2020.

[84] Mohamed Amine Ferrag, Leandros Maglaras, Sotiris Moschoyiannis, and Helge Janicke. Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. *Journal of Information Security and Applications*, 50:102419, February 2020.

[85] Daniel C Ferreira, Félix Iglesias Vázquez, Gernot Vormayr, Maximilian Bachl, and Tanja Zseby. A meta-analysis approach for feature selection in network traffic research. In *Proceedings of the Reproducibility Workshop*. ACM, 2017.

[86] S. Frankel and S. Krishnan. Ip security (ipsec) and internet key exchange (ike) document roadmap. Rfc, 2011.

[87] Pasi Fränti, Radu Mariescu-Istodor, and Caiming Zhong. Xnn graph. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 207–217. Springer, 2016.

[88] Pasi Fränti, Mohammad Rezaei, and Qinpei Zhao. Centroid index: cluster level similarity measure. *Pattern Recognition*, 47(9):3034–3045, 2014.

[89] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 2001.

[90] Bernd Fritzke. A growing neural gas network learns topologies. In *Advances in neural information processing systems*, pages 625–632, 1995.

[91] Ni Gao, Ling Gao, Quanli Gao, and Hai Wang. An intrusion detection model based on deep belief networks. In *2014 Second International Conference on Advanced Cloud and Big Data*. IEEE, 2014.

[92] Elizabeth Gardner. Maximum storage capacity in neural networks. *EPL (Europhysics Letters)*, 1987.

[93] Andrew Gelman and Jennifer Hill. *Data analysis using regression and multilevel/hierarchical models*. Cambridge university press, 2006.

[94] Jan-Mark Geusebroek, Gertjan J Burghouts, and Arnold WM Smeulders. The amsterdam library of object images. *International Journal of Computer Vision*, 61(1):103–112, 2005.

[95] Syed Ghazanfar, Faisal Hussain, Atiq Ur Rehman, Ubaid U Fayyaz, Farrukh Shahzad, and Ghalib A Shah. Iot-flock: An open-source framework for iot traffic generation. In *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*, pages 1–6. IEEE, 2020.

[96] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[97] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[98] Alex Graves. Adaptive Computation Time for Recurrent Neural Networks. *arXiv:1603.08983 [cs]*, February 2017. arXiv: 1603.08983.

[99] Jerzy W Grzymala-Busse. Rule induction. In *Data mining and knowledge discovery handbook*. Springer, 2005.

[100] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5769–5779, 2017.

[101] Liang Guo and Ibrahim Matta. The war between mice and elephants. In *Proceedings Ninth International Conference on Network Protocols. ICNP 2001*, pages 180–188. IEEE, 2001.

[102] Peter Hart. The condensed nearest neighbor rule (corresp.). *IEEE transactions on information theory*, 14(3):515–516, 1968.

[103] Alexander Hartl, Maximilian Bachl, Joachim Fabini, and Tanja Zseby. Explainability and adversarial robustness for rnns. In *2020 IEEE Sixth International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 148–156. IEEE, 2020.

[104] Alexander Hartl, Maximilian Bachl, Joachim Fabini, and Tanja Zseby. Explainability and adversarial robustness for rnns. In *2020 IEEE Sixth International Conference on Big Data Computing Service and Applications (BigDataService)*. IEEE, 2020.

[105] Alexander Hartl, Joachim Fabini, and Tanja Zseby. Separating flows in encrypted tunnel traffic. In *# PLACEHOLDER_PARENT_METADATA_VALUE#*, pages 609–616, 2022.

[106] Ramin Hasibi, Matin Shokri, and Mehdi Dehghan. Augmentation scheme for dealing with imbalanced network traffic classification using deep learning. *arXiv preprint arXiv:1901.00204*, 2019.

[107] Gaofeng He, Bingfeng Xu, Lu Zhang, and Haiting Zhu. Mobile app identification for encrypted network flows by traffic correlation. *International Journal of Distributed Sensor Networks*, 2018.

[108] Alexandre Heuillet, Fabien Couthouis, and Natalia Díaz-Rodríguez. Explainability in deep reinforcement learning. *Knowledge-Based Systems*, 214:106685, 2021.

[109] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[110] Nicolas Hohn and Darryl Veitch. Inverting sampled traffic. *IEEE/ACM Transactions on Networking*, 2006.

[111] J. Hu, J. Deng, and M. Sui. A new approach for decision tree based on principal component analysis. In *2009 Int. Conf. on Computational Intelligence and Software Engineering*, 2009.

[112] L. Hubert and P. Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[113] Jonathan Huggins, Trevor Campbell, and Tamara Broderick. Coresets for scalable bayesian logistic regression. In *Advances in Neural Information Processing Systems*, pages 4080–4088, 2016.

[114] Christian Igel and Michael Hüsken. Improving the rprop learning algorithm. In *Proceedings of the second international ICSC symposium on neural computation (NC 2000)*, volume 2000, pages 115–121. Citeseer, 2000.

[115] Félix Iglesias, Alexander Hartl, Tanja Zseby, and Arthur Zimek. Are network attacks outliers? a study of space representations and unsupervised algorithms. In *ECML-PKDD Workshops. Machine Learning for Cybersecurity (MLCS)*, 2019. publication pending.

[116] Félix Iglesias, Fares Meghdouri, Robert Annessi, and Tanja Zseby. Ccgen: Injecting covert channels into network traffic. *Security and Communication Networks*, 2022, 2022.

[117] Félix Iglesias and Tanja Zseby. Analysis of network traffic features for anomaly detection. *Machine Learning*, 2015.

[118] Félix Iglesias and Tanja Zseby. Time-activity footprints in ip traffic. *Computer Networks*, 2016.

[119] Félix Iglesias and Tanja Zseby. Pattern discovery in internet background radiation. *IEEE Transactions on Big Data*, 2017.

[120] Félix Iglesias, Tanja Zseby, Daniel Ferreira, and Arthur Zimek. Mdcgen: Multidimensional dataset generator for clustering. *Journal of Classification*, pages 1–20.

[121] Information and Irvine Computer Science, University of California. Kdd cup 1999 data.

[122] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[123] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[124] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[125] Steve TK Jan, Qingying Hao, Tianrui Hu, Jiameng Pu, Sonal Oswal, Gang Wang, and Bimal Viswanath. Throwing darts in the dark? detecting bots with limited data using neural data augmentation. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1190–1206. IEEE, 2020.

[126] Brian Alan Johnson, Ryutaro Tateishi, and Nguyen Thanh Hoan. A hybrid pansharpening approach and multiscale object-based image analysis for mapping diseased pine and oak trees. *International journal of remote sensing*, 34(20):6969–6982, 2013.

[127] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[128] Ismo Kärkkäinen and Pasi Fränti. *Dynamic local search algorithm for the clustering problem*. University of Joensuu, 2002.

[129] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[130] MJ Keith, A Jameson, W Van Straten, M Bailes, S Johnston, M Kramer, A Possenti, SD Bates, NDR Bhat, M Burgay, et al. The high time resolution universe pulsar survey–i. system configuration and initial discoveries. *Monthly Notices of the Royal Astronomical Society*, 409(2):619–627, 2010.

[131] John D Kelleher, Brian Mac Namee, and Aoife D'arcy. *Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies*. MIT press, 2020.

[132] Fabian Keller, Emmanuel Muller, and Klemens Bohm. Hics: High contrast subspaces for density-based outlier ranking. In *2012 IEEE 28th international conference on data engineering*, pages 1037–1048. IEEE, 2012.

[133] S Kent. Rfc 4303-ip encapsulating security payload (esp), 2005. *URL https://tools.ietf. org/html/rfc4303*, 2006.

[134] Farrukh Aslam Khan, Abdu Gumaei, Abdelouahid Derhab, and Amir Hussain. A novel two-stage deep learning model for efficient network intrusion detection. *IEEE Access*, 2019.

[135] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2(1):1–22, 2019.

[136] Hyang-Ah Kim and David R O'Hallaron. Counting network flows in real time. In *GLOBECOM'03. IEEE Global Telecommunications Conference (IEEE Cat. No. 03CH37489)*. IEEE, 2003.

[137] Seung Wook Kim, Yuhao Zhou, Jonah Philion, Antonio Torralba, and Sanja Fidler. Learning to simulate dynamic environments with gamegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1231–1240, 2020.

[138] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[139] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 2017.

[140] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[141] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

[142] Christopher Kruegel and Thomas Toth. Using decision trees to improve signature-based intrusion detection. In *International workshop on recent advances in intrusion detection*, pages 173–191. Springer, 2003.

[143] Xiao-hui Kuang, Jin Li, and Fei Xu. Network traffic generator based on distributed agent for large-scale network emulation environment. In *International Conference on Intelligent Science and Big Data Engineering*, pages 68–79. Springer, 2018.

[144] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 2015.

[145] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[146] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*. Springer, 2012.

[147] Sam Leroux, Steven Bohez, Elias De Coninck, Tim Verbelen, Bert Vankeirsbilck, Pieter Simoens, and Bart Dhoedt. The cascading neural network: building the Internet of Smart Things. *Knowledge and Information Systems*, 52(3):791–814, September 2017.

[148] Sam Leroux, Steven Bohez, Tim Verbelen, Bert Vankeirsbilck, Pieter Simoens, and Bart Dhoedt. Resource-constrained classification using a cascade of neural network layers. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, July 2015. ISSN: 2161-4407.

[149] Feng Li, Jae Won Chung, and Mark Claypool. Silhouette: Identifying youtube video flows from encrypted traffic. In *Proceedings of the 28th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2018.

[150] Zhipeng Li, Zheng Qin, Kai Huang, Xiao Yang, and Shuxiong Ye. Intrusion detection using convolutional neural networks for representation learning. In *International Conference on Neural Information Processing*. Springer, 2017.

[151] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 2017.

[152] Hyun-Kyo Lim, Ju-Bong Kim, Joo-Seong Heo, Kwihoon Kim, Yong-Geun Hong, and Youn-Hee Han. Packet-based network traffic classification using deep learning. In *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. IEEE, 2019.

[153] Yeon-sup Lim, Hyun-chul Kim, Jiwoong Jeong, Chong-kwon Kim, Ted Taekyoung Kwon, and Yanghee Choi. Internet traffic classification demystified: on the sources of the discriminative power. In *Proceedings of the 6th International COnference*. ACM, 2010.

[154] Zilong Lin, Yong Shi, and Zhi Xue. Idsgan: Generative adversarial networks for attack generation against intrusion detection. *arXiv preprint arXiv:1809.02077*, 2018.

[155] Lisa Liu, Gints Engelen, Timothy Lynar, Daryl Essam, and Wouter Joosen. Error prevalence in nids datasets: A case study on cic-ids-2017 and cse-cic-ids-2018. In *2022 IEEE Conference on Communications and Network Security (CNS)*, pages 254–262. IEEE, 2022.

[156] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing*, 2017.

[157] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing*, 2020.

[158] Mario Lucic, Matthew Faulkner, Andreas Krause, and Dan Feldman. Training gaussian mixture models at scale via coresets. *The Journal of Machine Learning Research*, pages 5885–5909, 2017.

[159] Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable ai for trees. *Nature Machine Intelligence*, 2020.

[160] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017.

[161] Andrew L. Maas. Rectifier Nonlinearities Improve Neural Network Acoustic Models, 2013.

[162] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*. Citeseer, 2013.

[163] Thomas Martinetz, Klaus Schulten, et al. A" neural-gas" network learns topologies. 1991.

[164] Johan Mazel, Romain Fontugne, and Kensuke Fukuda. A taxonomy of anomalies in backbone network traffic. In *Proceedings of 5th International Workshop on TRaffic Analysis and Characterization*, TRAC 2014, pages 30–36, 2014.

[165] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*. Elsevier, 1989.

[166] Fares Meghdouri, Maximilian Bachl, and Tanja Zseby. Eagernet: Early predictions of neural networks for computationally efficient intrusion detection. In *2020 4th Cyber Security in Networking Conference (CSNet)*, pages 1–7. IEEE, 2020.

[167] Fares Meghdouri, Félix Iglesias Vázquez, and Tanja Zseby. Modeling data with observers. *Intelligent Data Analysis*, 26(3):785–803, 2022.

[168] Fares Meghdouri, Thomas Schmied, Thomas Gärtner, and Tanja Zseby. Controllable network data balancing with gans. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021.

[169] Fares Meghdouri, Félix Iglesias Vázquez, and Tanja Zseby. Anomaly detection for mixed packet sequences. In *2020 IEEE 45th LCN Symposium on Emerging Topics in Networking (LCN Symposium)*, pages 120–130. IEEE, 2020.

[170] Fares Meghdouri, Félix Iglesias Vázquez, and Tanja Zseby. Cross-layer profiling of encrypted network data for anomaly detection. In *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 469–478. IEEE, 2020.

[171] Fares Meghdouri, Félix Iglesias Vázquez, and Tanja Zseby. Shedding light in the tunnel: Counting flows in encrypted network traffic. In *2021 International Conference on Data Mining Workshops (ICDMW)*, pages 798–804. IEEE, 2021.

[172] Fares Meghdouri, Tanja Zseby, and Félix Iglesias. Analysis of lightweight feature vectors for attack detection in network traffic. *Applied Sciences*, 2018.

[173] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[174] Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning*, pages 6950–6960. PMLR, 2020.

[175] Andrew Moore, Denis Zuev, and Michael Crogan. Discriminators for use in flow-based classification. Technical report, Queen Mary, University of London, Dept of Computer Science., 2005.

[176] Nour Moustafa and Jill Slay. 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, 2015.

[177] Nour Moustafa and Jill Slay. 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, November 2015.

[178] Nour Moustafa and Jill Slay. The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set. *Information Security Journal: A Global Perspective*, 2016.

[179] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[180] Khoi Khac Nguyen, Dinh Thai Hoang, Dusit Niyato, Ping Wang, Diep Nguyen, and Eryk Dutkiewicz. Cyberattack detection in mobile cloud computing: A deep learning approach. In *2018 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2018.

[181] Weili Nie, Nina Narodytska, and Ankit Patel. Relgan: Relational generative adversarial networks for text generation. In *International conference on learning representations*, 2018.

[182] Salman Niksefat, Parisa Kaghazgaran, and Babak Sadeghiyan. Privacy issues in intrusion detection systems: A taxonomy, survey and future directions. *Computer Science Review*, 25:69–78, 2017.

[183] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *International conference on machine learning*, pages 2642–2651. PMLR, 2017.

[184] Fannia Pacheco, Ernesto Exposito, Mathieu Gineste, Cedric Baudoin, and Jose Aguilar. Towards the deployment of machine learning solutions in network traffic classification: A systematic survey. *IEEE Communications Surveys & Tutorials*, 21(2):1988–2014, 2018.

[185] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*. PMLR, 2013.

[186] Lizhi Peng, Hongli Zhang, Bo Yang, and Yuehui Chen. Feature evaluation for early stage internet traffic identification. In *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2014.

[187] Sasanka Potluri and Christian Diedrich. Accelerated deep neural networks for enhanced intrusion detection system. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2016.

[188] Alun Preece. Asking 'why'in ai: Explainability of intelligent systems–perspectives and challenges. *Intelligent Systems in Accounting, Finance and Management*, 25(2):63–72, 2018.

[189] Donghong Qin, Jiahai Yang, Jiamian Wang, and Bin Zhang. Ip traffic classification based on machine learning. In *2011 IEEE 13th International Conference on Communication Technology*. IEEE, 2011.

[190] Qiaofeng Qin, Konstantinos Poularakis, Kin K Leung, and Leandros Tassiulas. Line-speed and scalable intrusion detection at the network edge via federated learning. In *2020 IFIP Networking Conference (Networking)*, pages 352–360. IEEE, 2020.

[191] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[192] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 427–438, 2000.

[193] Roger Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 1990.

[194] M Mazhar Rathore, Awais Ahmad, and Anand Paul. Real time intrusion detection system for ultra-high-speed big data environments. *The Journal of Supercomputing*, 72(9):3489–3510, 2016.

[195] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017.

[196] Russell Reed and Robert J MarksII. *Neural smithing: supervised learning in feedforward artificial neural networks.* Mit Press, 1999.

[197] E. Rescorla. The transport layer security (tls) protocol version 1.3. Rfc, 2018.

[198] E. Rescorla and N. Modadugu. Datagram transport layer security version 1.2. Rfc, 2012.

[199] Mohammad Rezaei and Pasi Fränti. Set matching measures for external cluster validity. *IEEE Transactions on Knowledge and Data Engineering*, 28(8):2173–2186, 2016.

[200] Shahbaz Rezaei and Xin Liu. Deep learning for encrypted traffic classification: An overview. *IEEE communications magazine*, 2019.

[201] Ronny Richardson and Max M North. Ransomware: Evolution, mitigation and prevention. *International Management Review*, 13(1):10, 2017.

[202] Maria Rigaki and Sebastian Garcia. Bringing a gan to a knife-fight: Adapting malware communication to avoid detection. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 70–75. IEEE, 2018.

[203] Markus Ring, Daniel Schlör, Dieter Landes, and Andreas Hotho. Flow-based network traffic generation using generative adversarial networks. *Computers & Security*, 82:156–172, 2019.

[204] Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, and Andreas Hotho. A survey of network-based intrusion detection data sets. *Computers & Security*, 86:147–167, 2019.

[205] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.

[206] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[207] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

[208] Yvan Saeys, Thomas Abeel, and Yves Van de Peer. Robust feature selection using ensemble feature selection techniques. In Walter Daelemans, Bart Goethals, and Katharina Morik, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 313–325, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[209] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 2015.

[210] Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. 1998.

[211] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[212] Phurivit Sangkatsanee, Naruemon Wattanapongsakorn, and Chalermpol Charnsripinyo. Practical real-time intrusion detection using machine learning approaches. *Computer Communications*, 34(18):2227–2235, 2011.

[213] Josep Sanjuas-Cuxart, Pere Barlet-Ros, and Josep Solé-Pareta. Counting flows over sliding windows in high speed networks. In *International Conference on Research in Networking*. Springer, 2009.

[214] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the burst: Remote identification of encrypted video streams. In *26th Security Symposium*, 2017.

[215] Minjoon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. Neural Speed Reading via Skim-RNN. In *ICLR*, March 2018.

[216] Syed Ali Raza Shah and Biju Issac. Performance comparison of intrusion detection systems and application of machine learning to snort system. *Future Generation Computer Systems*, 80:157–170, 2018.

[217] Khalid Shahbar and A Nur Zincir-Heywood. How far can we push flow analysis to identify encrypted anonymity network traffic? In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018.

[218] Mustafizur R Shahid, Gregory Blanc, Houda Jmila, Zonghua Zhang, and Hervé Debar. Generative deep learning for internet of things network traffic generation. In *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 70–79. IEEE, 2020.

[219] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. A detailed analysis of the cicids2017 data set. In *International Conference on Information Systems Security and Privacy*. Springer, 2018.

[220] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSP*, 2018.

[221] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSP*, pages 108–116, 2018.

[222] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017.

[223] George W Snedecor and William G Cochran. Statistical methods, eight edition. *Iowa state University press, Ames, Iowa*, 1191, 1989.

[224] Joel Sommers and Paul Barford. Self-configuring network traffic generation. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 68–81, 2004.

[225] CSL Sony and Kenjiro Cho. Traffic data repository at the wide project. In *Proceedings of USENIX 2000 Annual Technical Conference: FREENIX Track*, 2000.

[226] Murat Soysal and Ece Guran Schmidt. Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison. *Performance Evaluation*, 67(6):451–467, 2010.

[227] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 2014.

[228] Tuan A Tang, Lotfi Mhamdi, Des McLernon, Syed Ali Raza Zaidi, and Mounir Ghogho. Deep learning approach for network intrusion detection in software defined networking. In *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*. IEEE, 2016.

[229] Farhan Tariq and Shamim Baig. Machine learning based botnet detection in software defined networks. *International Journal of Security and Its Applications*, 2017.

[230] Taha AIT Tchakoucht and Mostafa Ezziyyani. Building a fast intrusion detection system for high-speed-networks: Probe and dos attacks detection. *Procedia Computer Science*, 127:521–530, 2018.

[231] Sergey Tulyakov, Ming-Yu Liu, Xiaodong Yang, and Jan Kautz. Mocogan: Decomposing motion and content for video generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1526–1535, 2018.

[232] Alfred Ultsch. Clustering with som: Uˆ* c. In *Proceedings of the workshop on self-organizing maps, 2005*, 2005.

[233] Félix Iglesias Vázquez, Tanja Zseby, and Arthur Zimek. Outlier detection based on low density models. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 970–979, 2018.

[234] Petr Velan, Milan Čermák, Pavel Čeleda, and Martin Drašar. A survey of methods for encrypted traffic classification and analysis. *International Journal of Network Management*, 2015.

[235] Eduardo Viegas, Altair Santin, Alysson Bessani, and Nuno Neves. Bigflow: Real-time and reliable anomaly-based intrusion detection for high-speed networks. *Future Generation Computer Systems*, 93:473–485, 2019.

[236] Rahul K. Vigneswaran, R. Vinayakumar, K.P. Soman, and Prabaharan Poornachandran. Evaluating Shallow and Deep Neural Networks for Network Intrusion Detection Systems in Cyber Security. In *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–6, July 2018.

[237] R Vinayakumar, KP Soman, and Prabaharan Poornachandran. Applying convolutional neural network for network intrusion detection. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2017.

[238] Ravi Vinayakumar, Mamoun Alazab, KP Soman, Prabaharan Poornachandran, Ameer Al-Nemrat, and Sitalakshmi Venkatraman. Deep learning approach for intelligent intrusion detection system. *IEEE Access*, 7:41525–41550, 2019.

[239] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004. Publisher: Springer.

[240] Kashi Venkatesh Vishwanath and Amin Vahdat. Swing: Realistic and responsive network traffic generation. *IEEE/ACM Transactions on Networking*, 17(3):712–725, 2009.

[241] Gernot Vormayr, Joachim Fabini, and Tanja Zseby. Why are my flows different? a tutorial on flow exporters. *IEEE Communications Surveys & Tutorials*, 2020.

[242] Joseph Wang, Kirill Trapeznikov, and Venkatesh Saligrama. Efficient Learning by Directed Acyclic Graph For Resource Constrained Prediction. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2152–2160. Curran Associates, Inc., 2015.

[243] W. Wang, Y. Sheng, J. Wang, X. Zeng, X. Ye, Y. Huang, and M. Zhu. Hast-ids: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection. *IEEE Access*, pages 1792–1806, 2018.

[244] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng. Malware traffic classification using convolutional neural network for representation learning. In *2017 International Conference on Information Networking (ICOIN)*. IEEE, 2017.

[245] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

[246] Nigel Williams, Sebastian Zander, and Grenville Armitage. A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification. *ACM SIGCOMM Computer Communication Review*, 2006.

[247] Jiajing Wu and Yongxiang Xia. Complex-network-inspired design of traffic generation patterns in communication networks. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 64(5):590–594, 2016.

[248] Kehe Wu, Zuge Chen, and Wei Li. A novel intrusion detection model for a massive network using convolutional neural networks. *IEEE Access*, 2018.

[249] Yihan Xiao, Cheng Xing, Taining Zhang, and Zhongkai Zhao. An intrusion detection model based on feature reduction and convolutional neural networks. *IEEE Access*, 2019.

[250] Keyulu Xu, Mozhi Zhang, Jingling Li, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. How neural networks extrapolate: From feedforward to graph neural networks. *arXiv preprint arXiv:2009.11848*, 2020.

[251] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional gan. *arXiv preprint arXiv:1907.00503*, 2019.

[252] Zhixiang (Eddie) Xu, Matt J. Kusner, Kilian Q. Weinberger, Minmin Chen, and Olivier Chapelle. Classifier Cascades and Trees for Minimizing Feature Evaluation Cost. *Journal of Machine Learning Research*, 15(62):2113–2144, 2014.

[253] Kenji Yamanishi, Jun-Ichi Takeuchi, Graham Williams, and Peter Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. *Data Mining and Knowledge Discovery*, 8(3):275–300, 2004.

[254] In-Kwon Yeo and Richard A Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 2000.

[255] Chuanlong Yin, Yuefei Zhu, Shengli Liu, Jinlong Fei, and Hetong Zhang. An enhancing framework for botnet detection using generative adversarial networks. In *2018 International Conference on Artificial Intelligence and Big Data (ICAIBD)*. IEEE, 2018.

[256] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, 2014.

[257] Adams Wei Yu, Hongrae Lee, and Quoc Le. Learning to Skim Text. In *ACL 2017*, pages 1880–1890, Vancouver, Canada, July 2017. ACL.

[258] Danni Yuan, Kaoru Ota, Mianxiong Dong, Xiaoyan Zhu, Tao Wu, Linjie Zhang, and Jianfeng Ma. Intrusion detection for smart home security based on data augmentation with edge computing. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2020.

[259] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[260] Harry Zhang. The optimality of naive bayes. *Aa*, 1(2):3, 2004.

[261] Junhui Zhang, Jiqiang Tang, Xu Zhang, Wen Ouyang, and Dongbin Wang. A survey of network traffic generation. 2015.

[262] Jianlong Zhou, Amir H Gandomi, Fang Chen, and Andreas Holzinger. Evaluating the quality of machine learning explanations: A survey on methods and metrics. *Electronics*, 10(5):593, 2021.

[263] Arthur Zimek, Matthew Gaudet, Ricardo J.G.B. Campello, and Jörg Sander. Subsampling for efficient and effective unsupervised outlier detection ensembles. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 428–436, 2013.